



UNIFORMING A TESTING ENVIRONMENT OF A MOBILE DEVICE

Heli Kontturi

Master's thesis
November 2012
Degree Programme in
Information Technology

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Information Technology

HELI KONTTURI:

Uniforming a Testing Environment of a Mobile Device

Master's thesis 78 pages, appendices 3 pages

November 2012

In the project I have been working together with my colleagues we have had difficulties with getting unified testing results with the mobile devices. The same device and the same phone software give varying results. That's why I decided to do some investigation why the results change so much. This is explained in the chapter "System under testing".

The most used special words are collected in Glossary chapter.

Chapter 1 Introduction introduces the actual work and the reason behind this thesis.

Chapter 2 explains testing theory. It is introduced first in general level. Then there is a sub chapter about manual testing which continues testing chapter with the focus on manual testing methods. The next sub chapter concerns test automation. This is included because the project of this thesis is connected with automatic testing project.

Chapter 3 introduces testing tools in general. There are manual and automated testing tools. Tools are not necessarily used in this project but they are otherwise known.

Chapter 4 tells about ATS, automated testing system. ATS has its own chapter because it is the system that was in use in my project.

Chapter 5 tells about Symbian and its history from the beginning. The history is included because of the Symbian's importance to the mobile world. The first Symbian smart phone is described in detail because it was the last Symbian phone ever, Nokia 808.

MeeGo is introduced in chapter 6 as a comparison to Symbian and in order to show that the history and importance is not yet as big as Symbian. N9 is selected as a comparison to Nokia 808. N9 was selected because it was the first and the last “MeeGo” phone Nokia produced.

Chapter 7 introduces the actual work. It tells about the problems and the methods we tried in solving them.

Key words: testing, mobile device, Symbian, MeeGo, automated test system,

TABLE OF CONTENTS

ABSTRACT

TABLE OF CONTENTS

GLOSSARY

1. INTRODUCTION	1
2. TESTING.....	2
2.1 Manual testing	11
2.2 Test automation.....	20
3. TESTING TOOLS IN GENERAL	24
4. ATS – AUTOMATED TESTING SYSTEM	29
5. SYMBIAN PHONES	34
6. MeeGo	46
7. SYSTEM UNDER TESTING	60
REFERENCES.....	67
APPENDICES	72

Appendix 1 FIGURE 17: Symbian kernel

Appendix 2 FIGURE 24: MeeGo domain view

Appendix 3 FIGURE 26: The MeeGo security Architecture

GLOSSARY

Apache Tomcat	Formerly known as Jakarta Tomcat is an open source web server, developed by Apache Software Foundation. It provides a "pure Java" HTTP web server environment for Java code to run. /41/
ATS	Automated Test System
ATS-worker	An application installed to a computer which takes care of the test runs. /1/
Automated testing	Testing using automated testing system.
Black-box testing	Internal functions of the program are not needed to know. The tester only knows the inputs and outputs of the program. /8/
Bug “ “error”, “bug”, ”fault”, ”mistake” and ”defect” are used as synonyms.” /5 p. 270/	
Exploratory Testing	Not predefined testing; tester does what a normal user would do.
Fault	“A fault is a defect in the program code which is the reason for a failure.” /5 p 270/
FPS-10/20/21	A device used to flash the software.
Fbus	Fast Bus cable is a standardized communication data cable for electronic hardware. Usually one piece acting as a master and the other one as a slave. /45/
Error	“Human function, which causes a defect.” /5 p. 270/
Gray-box testing	Combines the black box and white box techniques. The tester knows partially about the insides of the program and how it is supposed to work. /14/
Harness	Testing interface path
HTI	Harmonized Test Interface is an interface to a Symbian-based phone. It is used to control a test target. /1; 40/
HW-level	Hardware-level; testing the software “near” the hardware not with the normal user interface.
Linux	A free Unix like operating system for computers. /2/
Manual testing	Testing manually not automatically.
MeeGo	Nokia's and Intel's joint project. A free Linux based

	operating system, mainly for mobile systems. /3; 4/
Mobile phone, phone	Device under testing.
Mobile	See mobile phone.
MySQL	An open source database management system for relational databases. /42/
Perl	Interpreted, dynamic programming language (Practical Extraction and Reporting Language), originally developed in 1987. /44/
Phone software	Phone modem software under testing.
Quality Center	Web based testing and test management tool. /61/
Software A	Software used for production testing. Used to change “phone modes”. (Due to confidentiality matters, the name software A is used). /12/
Software B	Software that is “an adaptation” between modem software and phone user interface software. (Due to confidentiality matters, the name software B is used). /12/
STIF	Testing interface for Symbian non UI components. /1/
STAF	An open source framework used in testing, designed around the idea of reusable components, called services (The Software Testing Automation Framework). /43/
Symbian	An operating system designed for mobile phones.
Testing tools	Different tools used in testing.
Test case	It is a set of different conditions and variables to determine whether a software is working or not. Cases can be done automatically or manually. /6/
Test design techniques	Different techniques to derive test cases. /5 p. 273-274; 8; 9; 15; 16; 17/
UI	User interface
White-box testing	Uses the knowledge of the insides of the program in creating the test cases. /5 p. 273-274; 8; 9/

1. INTRODUCTION

The mobile testing project at work has caused a lot of problems with me and my colleagues. We have been having difficulties with getting unified testing results with the mobile devices. The results have been varying too much. That's why I decided to do some investigation about why the results change so much. Even though it is useful to have different kinds of systems to make sure that every possible scenario is tested, we were starting to get really frustrated and desperate about the results in the project. The software was tested with different computers and also by changing the different software in use. Software A was used for production testing and software B was an adaptation between the modem and the UI software. These are later in the text referred as software A and software B.

The idea of this thesis is to explain how the problem was approached and to find out the reasons behind the testing results. The best case would be to find some ways to make the results much more reliable and much more unified. It was needed, because it is necessary to trust that the results are actually correct and the software is working correctly as it should.

The tests included changing the software under testing to the previous software that had been known to have passed the tests or the general behaviour was really known. Normally the tests consisted of testing the basic HW-level phone functions inside the device. They included data transfer and phone calls with automatic testing, and also some manual test cases. Manual test cases consisted mainly of just checking some device data. They were not usually a problem at all. Normally testing was done a couple of times in a week.

This investigation could not jeopardize the actual testing process or have an effect for the everyday testing processes. No delays to reporting the results of the current works. This process was initiated by the members of the project at that time.

Before explaining how the actual work was done, I'll tell about testing and processes in general. Then I will explain something about the manual testing and automatic testing and then comparing their differences. And in addition something about the tools used in

testing, starting to focus much more on the current real testing work. First I'll tell about the tools that can be used in general testing and then much more in context with the current testing work.

ATS-system used in automation testing focuses on the automatic system which is used in my project for automatic testing.

After the theory, I'll tell about the mobile operating systems in two chapters. Symbian is chosen as one operating system because the mobile device used in this project is a Symbian device. MeeGo is chosen as a comparison because it is based on the free Linux software. The history of Symbian is told from the beginning as long version because of the importance of the OS to the mobile world. All the same things are tried to be told than in Symbian chapter.

Nokia 808 is introduced because it was the last Symbian phone. N9 is introduced because it was the first "MeeGo" phone ever and it is told to be the best phone.

At the end I'll tell about the actual testing idea of this thesis. Something about the different problems faced in different testing phases and the solutions that were attempted during testing. I'll also tell the conclusions if found.

2. TESTING

This chapter explains the theory behind testing and what is testing. First testing is explained more generally and if possible how it is connected to the actual work. Then the chapter divides into sub-chapters manual-testing and automatic testing.

What is testing then? We all know that testing is everywhere. Whether it is hardware testing or software testing, everything needs to be tested before it is taken in to use.

It seems like nowadays people take testing for granted. People complain that why does this not work. Or this has not been tested. There is no car or operating system that has not been tested. Or at least it should have been tested. But is testing so easy after all?

In everyday language testing means pretty much everything that can be tried out. People can say “I’ll test this out” but they do not really mean the actual testing. In software testing field it is defined as “planned fault searching”. It is done by executing software or parts of it. In this case, testing is normally done as trying to execute the program with some predefined data. The motivation behind this is just to prove that the program is working correctly. It is not about the amount of the bugs. And it is doing exactly what it is supposed to do. In other words, it does what it has been designed for. /5 p. 266; 7 p. 25/

Good testing includes: test planning, test cases, creation of test environment, test execution and the result reviewing. These phases take usually about half of the resources reserved for a software testing project. The other half should be reserved for the coding process. /5 p. 265/

When a new testing project is started, it starts from the beginning. Especially then, the planning phase is really important. That specifies how testing is supposed to go and it defines testable areas. This phase also includes test cases and how results are handled after the testing. /7 p. 25/

Normally a tester needs to check the results and make some kind of a summary for people who need to know them. In my project, there is no need for a tester to interpret the results. In this case, tester only needs to send an email about what was tested and what failed/passed. Most of the people probably have no idea of what test cases were actually tested. But still the results need to be unambiguous, so that those who need to know about them do not need to necessarily know exactly which cases were tested.

Is the testing done at some point so you can say that the software is now ready? The answer is, no. You cannot say the exact amount of testing that needs to be done. The amount of testing is not the same than the efficiency of testing. Sometimes it could be easier to do testing really fast but with careful planning rather than just try out something for days. /5 p. 266/

In Figure 1, there is a simple testing example. There is data that is given as input, which is then executed with the inputted data. With these values, the program shows result

“Y”. Inserted data can be anything outside the system and respectively all the results can be any actions outside the system.

The testing is correctively done if the result is correct and that the inner state has been correctively changed to another state. Therefore to successfully complete the testing a tester needs to know what data is inserted and what should be the final result. The specifications are/should be designed for that purpose. /5 p. 266-267/

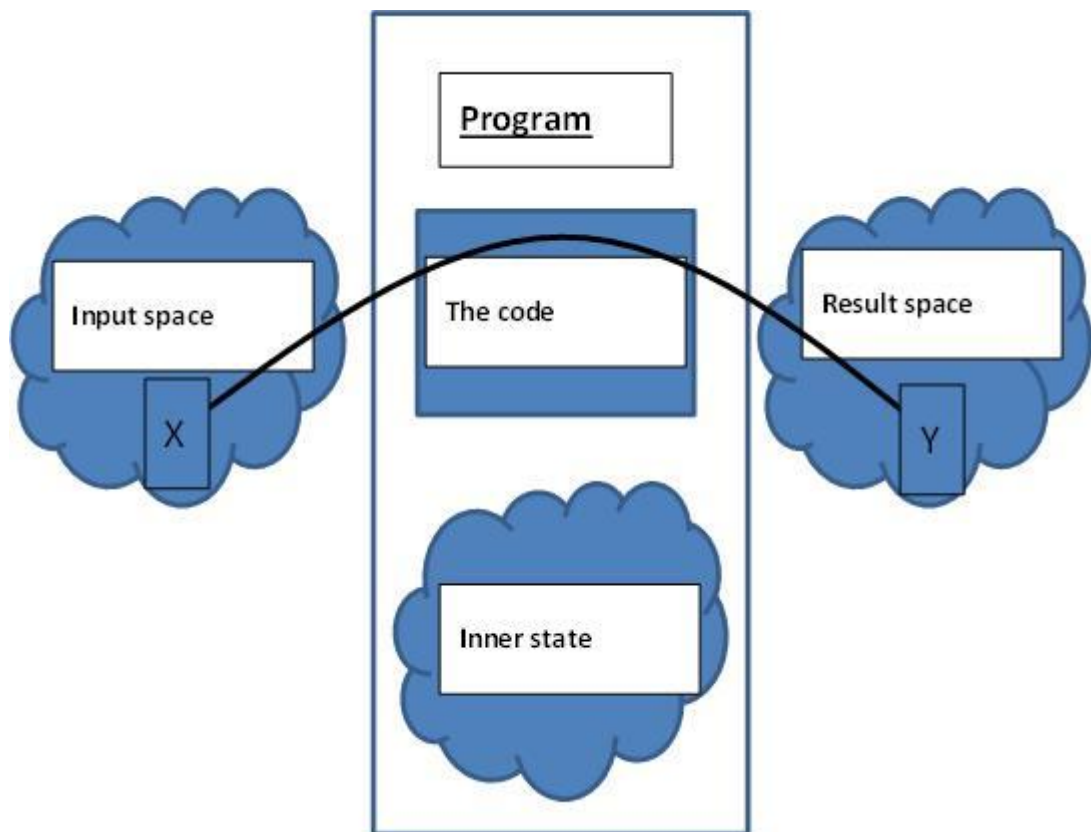


FIGURE 1: Software testing /7 p. 18/

In testing language error, mistake, and bug are usually thought to mean the same thing. An error is a deviation from the specification. The software is doing something that it is not supposed to do according to the specification. A program can work really slowly or user has troubles in using it. Errors can be really annoying or they can prevent the user from using the program. In the worst scenario the errors can even cause something to break. /7 p. 26/

According to one interpretation by Ilkka Haikala and Jukka Märijärvi, “an error is a human function, which causes a defect. A fault is a defect in the program code which is the reason for a failure.” /5 p. 270/

Really “sufficient” testing coverage is gained when the inputted data consists of all the possible correct and incorrect values. In reality that kind of testing coverage is practically impossible to gain. For example, the inner state of the program has an effect to the final results and how the data is executed. Different states in the program can cause the program act differently. That can make the results different even with the same data. /5 p. 266-267/

V-model of testing, (Figure 2) is a development model which describes the relationships between each phase of development life cycle in software testing. Left side of the Figure consists of development phases and right side contains testing phases. For every testing level, there is a correspondent planning level. For example, in acceptance testing the planning is done at the requirement phase and in system testing during specification phase and so on. Every phase can be tested as soon as it is done from the development side (left side of the model).

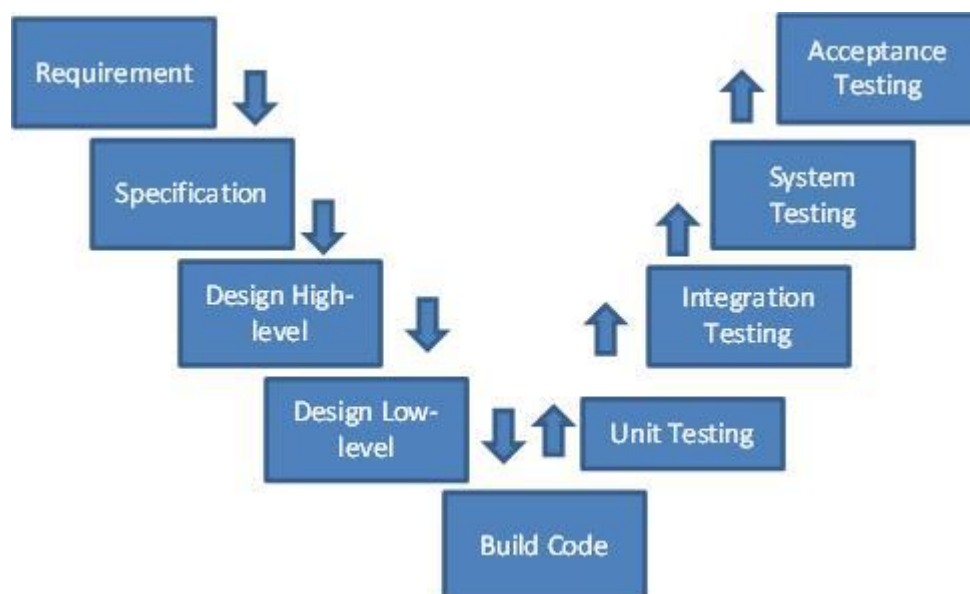


FIGURE 2: V-model of testing /10/

When we look at the Figure 2, which shows the testing V-model, we can see that it starts from Requirements Phase and ends with Acceptance Testing Phase. A mistake in requirements found at the acceptance phase would cost a lot more to fix than if it had been found earlier. Just because the error was in the requirements and it was not tested thoroughly in that phase, everything needs to be done again. That is called regression testing when everything is tested again, in spite of the testing level. That includes finding the requirements again because something was done wrong in that phase. Many

think that testing should happen only in testing phases, but clearly testing should be done after every phase. /5 p. 272; 11/

Testing proves that there are faults or bugs in the software. However, nothing proves that the software is flawless. Testing cannot cover all the possible and impossible situations. Software should still be tested to make sure that the most serious bugs are fixed. /7 p. 26/

It is evaluated that there is approximately one error for every few dozen lines of code. Even in software that has been in use for a long time. Approximately 5% of the bugs are not found. And that is because one state can cause one result with the same data and different states with different data can cause very different result combined together. Or one error can “fix” the other error. /5 p. 269-270/

There are four levels in testing, which can be seen in the Figure 3 below.

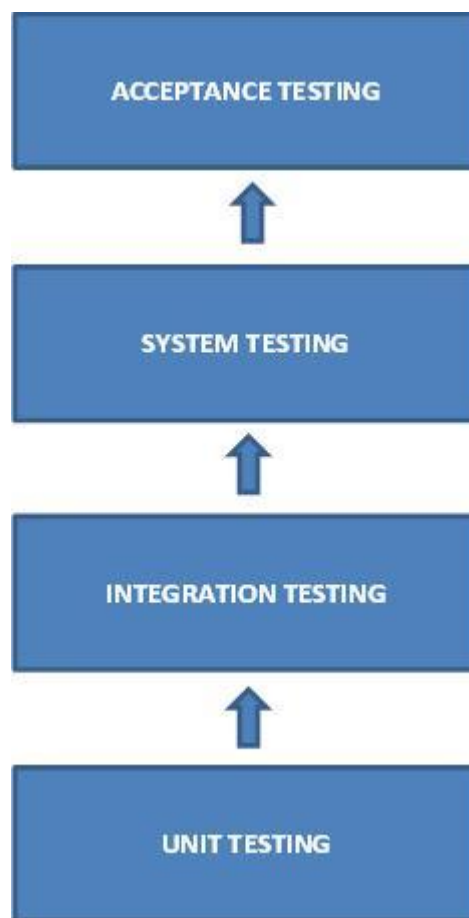


FIGURE 3: Software Testing Levels /13/

In unit testing an individual unit or component is tested. The idea is to make sure that the single module is working correctly as designed. Testing is usually carried out by the module/unit implementer. Testing environment can be some PC which resembles real time system. Test beds can be used to simulate the functionality of the unit/module. /5 p. 270-271; 13/

When all single modules are combined together and tested, it is called integration testing. The idea behind the integration testing is to find the errors between the integrated units. This phase also makes sure that the units work correctly together. Testing can be carried out alongside with unit testing. Integration testing is proceeding normally bottom up direction. This means that testing is starting from the low level modules all the way to top. In top down testing it goes from the other way down. /5 p. 272; 13/

System testing phase tests the different parts of a system (which consists of modules / units) which has been integrated to a complete system. The idea is to make sure that the whole system is complying with defined requirements. System testing tests also the non-functional aspects of the system, like performance, reliability and usability. The results are compared to functional documents and customer specific documents. Whoever does system testing must be as independent from development people as possible. /5 p. 272; 13/

System testing is followed by acceptance testing. It can be done together with system testing or field testing. This testing phase evaluates the system against the requirements and decides if it is ready for delivery. /5 p. 272; 13/

There are two basic ways of selecting test cases. They are called white box testing and black box testing. In black box testing internal functions of the program are not known. The tester only knows the inputs and outputs of the program (Figure 4).

Specifications and other internal knowledge about the program are needed for test case creation. They can test either functional or non-functional aspects of the system. Normally designer selects valid and invalid inputs and checks the correct outputs for them. /5 p. 273-274; 8; 9/



FIGURE 4: Black box testing /8/

Black box testing typically uses following test design techniques:

- Equivalence partitioning: Test cases are divided into equivalence classes. One value is tested from each of the classes. If test case works with one value in the class, it is assumed that it works with every value inside that class.
- Decision table testing: Each decision has different condition variables which lead to different actions.
- All-pairs testing: Tests all the possible combinations of each paired inputs.
- State transition tables: Shows the current and the next states of the program. It is a truth table which has some inputs as current state and some outputs next state.
- Boundary value analysis: The values between the equivalence classes are chosen as test cases. When moving towards the top in V-model, testing is usually changed to black box testing. /5 p. 273-274; 8; 9; 15; 16; 17/

White box testing uses the knowledge of the insides of the program. A tester uses this knowledge when creating test cases (Figure 5). Since the program code is known, the tester selects some inputs and finds out the outputs. Testing can be applied at many levels; it is normally done at the unit level. It can be used to test paths between different units or within the sub-systems. With the help of this method, lots of errors can be found. But it can't always detect errors caused by unimplemented parts or faulty requirements. This goes back to the requirements phase, where the mistake was actually done. /5 p. 273-274; 8; 9/

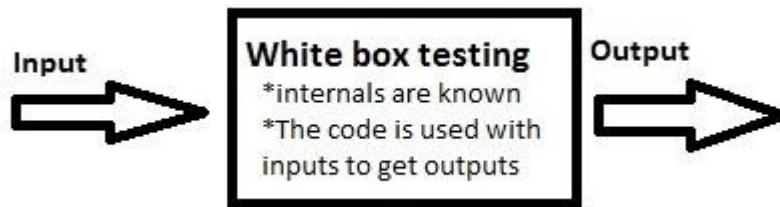


FIGURE 5: White box testing

White box testing typically uses the following test design techniques:

- | | |
|-----------------------|--|
| Control flow testing: | The order in which the statements are executed or evaluated. When a statement is executed it can lead to different choices which make up the paths being followed. |
| Data flow testing: | Focuses more on the variables. Tester can change the variables throughout the program and observe the effects. It is also related to path testing. |
| Branch testing: | Executes every branch decision at least once. |
| Path testing: | Selected paths throughout the program are executed at least once. This helps checking for the forgotten code and the orders of the executions. |
| Statement Coverage: | Describes how much code has been tested. |
| Decision Coverage: | Is also known as branch coverage. Two or more choices in a statement are known as decision, because it has two different paths. /9; 18; 19; 20; 21; 22; 23/ |

There is also a technique known as grey box testing. It uses the knowledge about the black box and white box techniques therefore it is a combination of the two techniques. The technique is used for collecting information for defining test cases. The tester knows partially about the insides of the program and how it is supposed to work. Testers require much more detailed descriptions of the documents and the applications. /14/

Gray-box testing techniques are:

- | | |
|-----------------|--|
| Matrix Testing: | The developers define variables that are used in the program with the different risk associated with them (business or technical). Matrix testing makes sure that requirements are |
|-----------------|--|

covered in testing. Testers can derive test cases directly from the matrixes.

Regression Testing: Testing is done after making a functional change into the system. To make sure that the change did not make the software to regress.

Pattern Testing: Analyzing historical data and defect of the previous system. The idea is to find out the reason why the failure happened by looking at the code. It uses predefined common test patterns for test design. Patterns are cost effective for test case creation because older patterns can be modified and reused.

Orthogonal array testing: Statistical testing technique is used to reduce the combinations and provide maximum coverage with a minimum number of test cases. An example system with 4 parameters takes 4 different inputs as values. To test all the combinations, tester would need 256 different test cases. Using a subset of the combinations, test cases can be minimized. All the possible pair combinations occur only once in the array. It is good for testing complex applications.

/14; 24/

Gray-box testing is good for web applications. White box testing cannot be used because there is no source code available. It can help to confirm that the software is meeting the correct requirements. /14/

At the end of testing, tester should have lots of different documents (Figure 6). When testing is planned correctly, there should be testing specification document which defines testing. There can be system testing plan, integration test plan or unit testing plan with every single test. And in addition a test report for the executed test cases with test statuses. In really small projects it is usually enough that there is only one test plan which covers all the testing. /5 p.281-282/

Test plan covers what areas are tested, when are they tested and what are the expected and needed results. Also end criteria for testing must be defined in the document. Errors

found in different levels/phases should be reported and documented for possible future learning. /5 p.281-282/

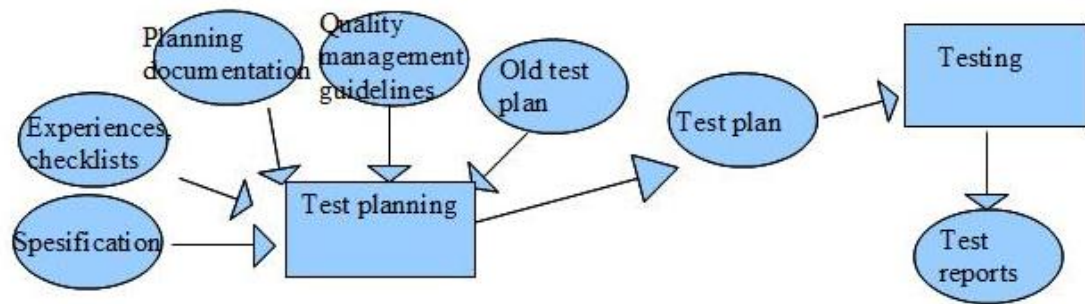


FIGURE 6: Testing planning /5p. 280/

2.1 Manual testing

When talking about testing, people usually assume that it means manual testing. This chapter continues about testing theory started in the previous chapter, but more from the perspective of a manual tester. The next chapter describes automated testing. Because my project involves manual testing with a mobile device, that is used as an example when needed. That does not mean that there cannot be other kind of manual testing.

The tester is in the role of a real user. The manual tester tries to do what a normal user would do, good or bad. The tester should always follow the test plan that has the information about testing and about the coverage of testing. It should be written before the testing starts. /25/

Exploratory testing is usually done manually. Sometimes it is all the testing that is needed. In that case the tester does not have to follow some exact procedure. Sometimes there could be more planned test cases to be followed but normally exploratory testing explores the interface (user interface testing) and goes through all the possible features with the help of the earlier tests and their results. For exploratory testing being successful, testers need to know their own areas very well. /25/

In addition to exploratory testing, one common testing form is usability testing. It uses user-centric interaction to explore the device/software functions by testing it with normal user point of view. This is one really important testing method because usually

most of the test cases follow certain predefined paths. According to the Wikipedia article about usability testing, “Usability testing measures the usability, or ease of use, of a specific object or set of objects, whereas general human-computer interaction studies attempt to formulate universal principles.” /33/

Usability testing is another form of black box testing technique. The idea is to find out how people are using the product/software in order to find the errors. It measures response in four different areas: efficiency, accuracy, recall and emotional response. Usability requirements must be defined beforehand (also usability goals). If the goals are not met after the testing then the software/device is not considered usable. The following items are measured:

1. Efficiency
 - How long does it take to complete certain tasks?
2. Accuracy
 - How many mistakes were done while completing the tasks?
3. Recall
 - How much is remembered immediately afterwards or much later?
4. Emotional response
 - What was the user’s impression? Would the user recommend this system? /33/

Systems needed to be usability tested can also be located in remote locations. This is where remote usability testing comes into the picture. Remote testing can be synchronous or asynchronous. In synchronous testing, a video conferencing or some remote application sharing tool, like WebEx, is normally used. Asynchronous testing uses more face-to-face communications. Asynchronous method can include automatic gathering of user inputs, for example button clicks or user logins. One more general usability testing method is expert review. It relies heavily on the experts experiences in evaluation. /33/

Most of the projects in engineering world use much more rigorous methods when manual testing is used to find as many defects as possible.

Planned and controlled testing follows predefined test cases and goes normally according to the next steps: /25/

1. High level test plan is used, where all the resources are listed.
2. Test cases are detailed, and the steps are accurate, with expected results.
3. Test cases are assigned to a tester who follows the steps exactly and saves the results.
4. A written test report is done of all the findings. /25/

Usually in a large project, much more strict approach to testing is used / needed. They follow normally the waterfall model (Figure 7). Some studies show that there are no really big differences in finding the bugs with the projects which follow the waterfall model. According to Wikipedia article about manual testing, "At least one recent study did not show a dramatic difference in defect detection efficiency between exploratory testing and test case based testing." This means that it really does not matter which testing method is used, as long as the testing is defined correctly from the beginning until the end. Correctly defined testing also includes selecting correct test cases.

As discussed in the previous chapter, manual-testing can be black box, white box or gray box testing (See chapter 2). /25/

Testing can also use dynamic- and static approaches. In dynamic testing the tester is actually running the software. It analyzes the physical response of the software or the system under testing. In other words, software must always be compiled and run. The tester/developer gives different inputs and examines the outputs against the correct results. Dynamic testing is always based on the specifications where the test cases and other criteria are defined. Dynamic testing methods include unit tests, integration tests, system tests and acceptance tests. /29/

Static testing is not as detailed testing as dynamic testing. The software is not actually used. Dynamic testing technique is mainly used for verifying the sanity of the code, algorithm, or document. Static testing is more used as checking the code and trying to find bugs without running the software. Mainly the developers are using the static testing method for reviewing the code before compiling it. Static testing can also be automated. The interpreter examines the code syntax for validity. /30/

In dynamic testing, the software under testing is executed, while static testing verifies the documents and the sanity of the code. Software is actually never executed in static testing. /25/

If we go further down the testing path, we find terms “functional” and “non-functional” testing. Functional testing means that the tester is checking the actual functions of the device/software. For example in mobile device testing, the functional tester can give input and the application should give appropriate output. /25/

Non-functional testing checks the software for a non-functional requirements, like performance (reliability, scalability), compatibility and fitness of the system under test, its security, usability, stress testing, endurance, recovery and volume testing /25; 32/

Functional testing verifies the software functions against the requirements and specifications.

Functional testing typically involves the following steps:

1. Finding the functions that the system should perform according to the specifications.
2. Inputs are created based on the specifications.
3. Outputs are determined based on the specifications.
4. Test cases are executed.
5. The actual results from testing and the expected results (specifications) are compared. /31/

Testing usually follows the waterfall model, it is “flowing” downwards (Figure 7) through the testing phases. The model starts from the requirements and goes through design, implementation, and verification to maintenance. /26/

The waterfall model comes originally from manufacturing and construction industries. At the time, there was no formal methodology so this was adapted to software development too. The first formal waterfall model was cited in 1970 by Winston W. Royce. It was presented as an example of a flawed and non-working model. /26/

According to the model, one should proceed to the next phase only when the previous one is finished. There are a lot of variations of the model. /26/

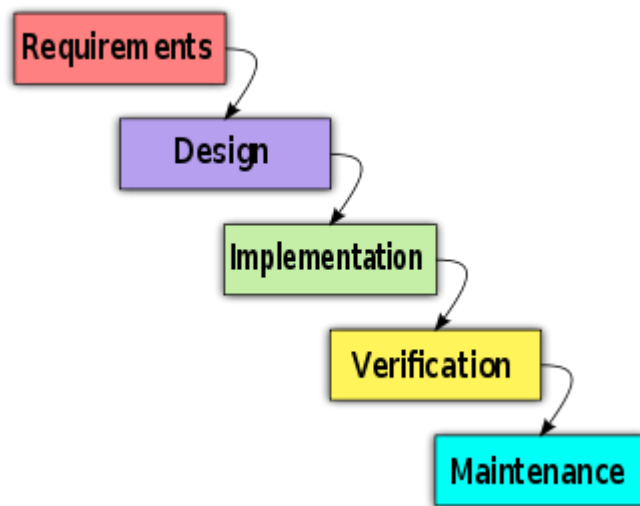


FIGURE 7: Waterfall model /26/

In Royce's original waterfall model, the phases are in the following order:

1. Requirements specification
2. Design
3. Construction (implementation)
4. Integration
5. Testing and debugging (validation or verification)
6. Installation
7. Maintenance. /26/

Agile testing can be used either in manual or in automated testing project (Figure 8). It is included in this chapter because I personally have never been in agile automated testing project but only in manual testing project. According to the Wikipedia article about Agile software development, Agile testing in software development is “a group of methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. Agile way of testing encourages among other things, adaptive planning and rapid and flexible response to changes.” Basically this means that different teams are working together locally and remotely. They must be ready to change the requirements frequently if needed. /27/

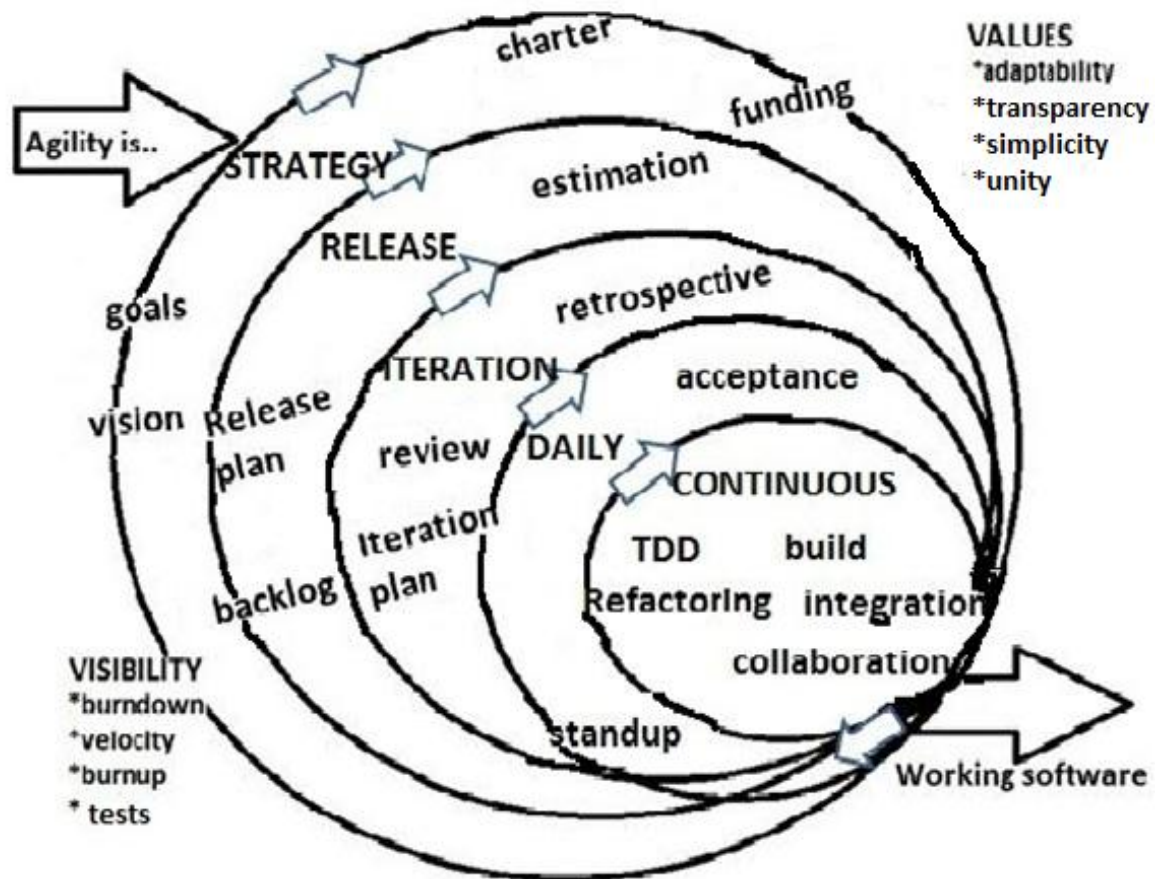


FIGURE 8: Description of Agile testing /27 modified/

The Agile Manifesto was published by the developers to define the approach method, now known as the Agile development.

Agile manifesto is consisted of 4 principles:

1. Individuals and interactions: The team is self-organized and motivated.
2. Working software: Show the customer working software instead of pile of documents.
3. Customer collaboration: Requirements change during the project, customer involvement is important.
4. Responding to change: Focus is on quick responses to changed requirements (principle 3) and thus continuous software development. /27/

Twelve principles behind the Agile way of working in the Manifesto include:

1. Customer satisfaction: achieved by rapid delivery
2. Changing the requirements: always welcomed
3. Working software: delivered frequently
4. Working software: the principal measure of progress

5. Sustainable development
6. Close, daily co-operation between business people and developers
7. Face-to-face conversations: very important and they are encouraged
8. Projects are built around motivated individuals
9. Continuous attention: needs to be directed to technical excellence and good design
10. Simplicity is emphasized
11. Teams are coordinated and ordered by themselves locally: Self-organizing teams
12. Regular adaptation to changing circumstances: Requirements can be changed rapidly. /27/

Agile method is suitable when the criticality is low, the team is small and the developers are experienced, requirements change often, and the culture is responding to the changes. Plan-driven development method is suitable when the criticality is high, the team is large with junior developers, requirements do not change very often, and the culture demands order. Formal method is suitable when the criticality is extremely high, team is consisted of senior developers, requirements are limited, and extreme quality is the main goal. /27/

Agile way of working includes many specific methods. Most of them encourage rapid changes, teamwork, adaptability and development. Larger tasks are broken into small tasks and there is no need for “big” planning. The tasks are usually done in small phases. One phase is normally from one to four weeks long. Each phase or iteration consists of a team working together in all of the functions: planning, requirements analysis, design, coding, unit testing and acceptance testing. At the end of the each one to four week phase, a working product is demonstrated. /27/

Methods in Agile projects emphasize on face-to-face communications. This is possible when the whole team is in the same location. Otherwise a videoconferencing, email or instant messaging can be used. There are usually five to nine people working in an Agile team. If the team is much larger, there might be communication problems among other things. Team meetings or scrums or daily stand-ups, should not take more than 15 minutes. Everybody should tell what they have been up to since the last meeting very briefly. Standing up usually limits the time effectively.

Agile way is not recommended for cultures that are not working without constant supervision. Then they will need more planned methodology. /27/

According to the Wikipedia article about Agile development, some of the methods used in Agile projects are:

1. Agile Modeling
2. Extreme Programming (XP)
3. Feature Driven Development (FDD)
4. Kanban (used in development)
5. Lean software development
6. Open Unified Process (OpenUP)
7. Scrum. /27/

Agile project uses also some specific tools and techniques, such as continuous integration, automated or xUnit testing, pair programming, test-driven development, design patterns, domain- driven design, and code refactoring. When Agile model is compared with waterfall model, we notice that in waterfall model, development finishes first and testing is done after that. In Agile model, testing is done simultaneously with the development. /27/

To conclude manual-testing chapter, testing is a process which aims that the software/device:

1. Is done according to the requirements defined earlier.
2. It is working according to the specifications (as expected).
3. Can be re-implemented later with the similar way.
4. Is approved by all the parties involved, including users. /35/

These hold true in all kinds of testing, whether it is manual-testing or automated testing. Testing can be implemented at any time in the process. Most the testing is happening after the requirements have been established and the development has been completed. When using Agile methods, testing process is constantly on-going together with the development. /35/

When using different development methods, the purpose and the path how to get the results vary. Different methods take different focus on the whole process. Agile process

focuses on the test driven development and the developer has much more responsibility on testing before more formal testing is done by the actual test team. The more formal procedure starts after the development is finished. /35/

Testers can do sometimes too much useless work. They should always be aware of that all the bugs or mistakes can never be discovered. Some of the bugs can appear after the other one has been fixed. They can be largely reduced with good planning beforehand but there is always something which may or may not affect the function or be visible to the user. There are different kinds of approaches to testing and it can be done in different levels depending on test objectives (See Figure 3). /35/

There are lots of variations in test cycles depending on the company and the way the cycles have been adapted. When companies are testing according to the waterfall model, it could look like this example. This example has 9 different phases.

1. Requirements analysis: Testing begins when testers and developers are defining the correct requirements. The requirements need to be correct in order to define the correct tests.
2. Test planning: Produces a document which describes how the testing should be done.
3. Test development: Defines what kind of procedures and scenarios are used in testing.
4. Test execution: Testing should be done according to the test plan.
5. Test reporting: Testing should produce a test report, which defines the fate of the software.
6. Test result analysis: The developers are analyzing the defects found in testing. The defects are prioritized. It is decided which defects are going to be fixed at this point.
7. Defect retesting: A fixed defect needs to be retested to make sure that the fixes are working.
8. Regression testing: The software needs to be tested again for any regression.
9. Test closure: When the testing meets the exit criteria, all the documents are archived for future use. /35/

2.2 Test automation

Automated testing or test automation can ease up testing some times. Test automation is using special software to control test execution. It can be much less laborious work than manual testing and much easier to find certain defects. It can reduce the load of a tester significantly. The tester can leave automated testing system running tests overnight and check the results next morning instead of spending the whole night waiting for long tests to be finished. A computer can follow predefined test scripts very precisely. Test scripts must be created manually and they must be correctly done. Otherwise the automation program is not able to execute them. Creating the tests is the bigger work to be done. Executing is quick and can be done repeatedly. However, interpreting test results, especially very large files, can be extremely time consuming and must usually be done manually. In normal cases, test automation is used together with manual testing. /25; 28/

Normally an automation tester needs to have a basic coding knowledge. The test cases are written in code form for the program. There is a way of creating cases automatically, a model based testing. Sometimes it can help non-technical people by creating test cases in plain English. /28/

Before testing, a tester must always think if manual testing makes sense. Testing large amounts of users practically requires automated testing. Different drivers and large libraries are really hard to test manually; therefore they need be tested using automated testing. All kinds of graphical user interfaces (UI) or screens where something visual is evaluated, are easier to test manually. There are some testing frameworks which can be used for user interface testing. They record keystrokes and different gestures. These tests can give “wrong” for different inputs. That happens if some key press is a little bit different and the test may not recognise it. /25/

There are two approaches to test automation; Code-driven testing and Graphical user interface testing. In Code-driven testing, modules and libraries are tested with many inputs to make sure that the outputs are correct. Graphical user Interface testing framework generates different user events, like mouse gestures and clicks and validates that the behavior is normal. Graphical user Interface tester is testing the interface to ensure it is meeting the specifications. /28; 34/

In manual testing there is a usability testing method called expert review. An automated version of that is naturally automated expert review. It is done by creating a program with certain rules to be evaluated in usability testing. Automated expert review does not provide as good and insightful information than an expert human. However it can be done quickly and they can be more consistent. There have been ideas of creating made up users for automated testing, which can be a step towards Artificial Intelligence community. /33/

Using testing frameworks is becoming really popular in automation testing. Popular frameworks are xUnit, Junit and Nunit. They allow running unit tests in order to make sure that the code is behaving correctly under different circumstances. That is called “Code Driven Test Automation”. It is a one key method in Agile development where it is known as “Test Driven Development”. / 28/

All needed unit tests are written before the code is done. If the tests are failing, then the code is not complete. That is why the code produced is much more reliable and much more covered than manually explored code. The code is tested along with the development according to the Agile model. The defects are found in the early phase when it is inexpensive to fix them. And it is easier to change the code to another form when it is much simpler. /28/

Test Automation tools provide good features for user action, gesture recording and playing, and in addition to comparing them to the expected results. This approach does not require too much development. Instead it can be used with all the applications with graphical user interface. However, these features might not be so reliable after all. If something is moved or renamed, the gesture may require test re-recording and that might add irrelevant actions to the tests. /28/

Web site testing tool is using entirely different techniques. It does not monitor actions or window events, it is reading HTML code. Using that kind of tool a tester does not require much development skills, if at all. Another way to do test automation is to build a model of the application under testing. A user can execute the tests just by modifying the parameters. No scripting or coding skills are needed. Drawback about this model is that the testing scripts are still used to maintain the model. When there are changes to the application, the scripts need to be modified also. /28/

Graphical testing requires also good set of test cases. The cases must cover all the possible functionality and especially the interface itself. Testing the GUI is a really demanding task because in the graphical interface there are lots of different operations to be tested. There is also a problem that some functions need to be done in an exact sequence, which can also be a problem. The more complex the sequencing is the more operations it needs to be accomplished. This is also increasing the size of the test cases. Regression testing with graphical interfaces is a demanding task. The interface can be changed over the different versions. Some items may change places in different versions which also makes the testing paths different than originally. Because of the problems mentioned earlier, test automation has gained a lot of recognition in testing graphical user interfaces. /34/

Another newer approach involves a planning system. Problems are solved with using four different parameters. According to the Wikipedia article about graphical user interface testing, the parameters are:

1. A preliminary or initial state: The state before starting.
2. A goal state: The final state where testing leads to.
3. A set of operators: They are used together with the set of objects.
4. A set of objects: The set of operators are used to operate them. /34/

The system is using the four parameters. It determines the correct path starting from the zero state, proceeds to the initial state and all the way to the goal. The system is using sets of operators and objects to operate on. In a planning system called “IPP”, the system is firstly analyzed for the possible operations to be found. Then the initial state and the goal are defined. Finally the system is determining the path from the defined initial state to the goal. This has become the test plan. /34/

The planning system creates lots of solutions to a tester's problems. According to the Wikipedia article about graphical user interface testing:

1. The test plans are always valid: There is a correct test plan which uses the correct operators or there is no plan at all.
2. A planning system pays attention to order: When the operations need to be done in a specific order, test cases can be really problematic. When the cases are done manually it can lead to lot of errors.

3. A planning system is goal oriented: The tester needs to focus on the functionality of the product/software. /34/

When the tests are done manually, the testers are more concentrated on how to test some function. While the tester is using the planning system, he/she can be more focused on what functions to test. Sometimes the system can find the new paths that were not seen by the tester. One interesting test case creation method would use a novice tester for getting a good coverage. Normally an expert has a certain and maybe too predictable and direct way of doing things. Novice would probably take a much longer path to achieve the goal therefore that would test more. On the other hand a novice would eventually become “the expert” and the paths would not be chosen the same way anymore. /34/

Graphical User Interface testing strategies come from the command line testing. “Capture/Playback” is a very commonly used strategy. It is “a system where the system screen is captured”. After capturing, the bitmapped image can be played back and the tester can compare the captured screen image to the expected result. The image is either right or wrong therefore it would be easy to automate the kinds of tests. /34/

Selenium framework is used in browser automation and it is mostly used for web application testing purposes. With a browser add-on it is possible to record-and-playback many kinds of interactions with the browser. /52/

There can still be lots of problems when capture/playback is used with the graphical user interface. The screen might look different in different situations or some window or button is slightly moved to other direction. However, the system is still the same. It only looks a little different, which makes the automated testing hard. The automated system does not know that everything is ok even though the fonts or sizes are slightly changed. This problem has been solved by not recording the actual appearance but the events behind the screen. Another way to test is to use a driver which sends and receives events from the other program to the one under testing. /34/

Many are relying more on the automated scripting techniques. Automated systems are not humans and they cannot do everything humans can do in testing. The scripts still need to be written and they need to be correct for them to be useful. Especially regression testing should be as automated as possible. /25/

There are lots of tools designed for test automation, different test generators, comparators and coverage tools. The tools are explained in more detail in the next chapter. /5 p. 278-281/

Always important question is, what makes sense to be automated and when. All the testing does not need to be automated. Choosing the correct features to be automated defined the success of the automation. Testing unstable features and the features that are changing should be avoided. /28/

3. TESTING TOOLS IN GENERAL

This chapter deals about the testing tools, both manual and automation tools. The focus is more on the automation side. The tools introduced here are not necessarily used in my project, but they are generally known. This chapter excludes the ATS system because it is explained in the next chapter.

One manual testing program is called Phoenix Service Software. The program is used for flashing the mobile phone and doing some tests. /49/

Testing can be made easier by using testing tools and debuggers. They can be used for monitoring, getting code coverage reports, benchmarks and performance analysis. There are even tools for automated functional graphical user interface testing. /35/

Testing should be as much automated as possible, especially in regression testing. Otherwise testing can be really demanding. For example in mobile device testing, repeating phone calls 50 or as much as 100 times can be really boring and tiring. There are many tools to make testing more easier to automate. There are test bed generators, test case generators, comparators and coverage analysers. /5 p. 278- 281/

Test bed generators are used for creating a test bed for a testable program. The generators are good for testing functions or classes in an isolated “clean” environment. It is also possible to automate verifying the results. Wanted results are described to test bed generator. In system testing, it is possible to record inputs and use them again when needed. /5 p. 278- 281/

Comparators can be used to compare the test result for the previously run tests. The problem with these can be changing data for example the time and date. /5 p. 278- 281/

Test coverage analyser measures the test coverage with some previously described method while executing the code. Analyser can measure the execution times and processor usage. This makes the code bigger and slows down the system. That is why test coverage tool usage is limited in real time and system testing. Example of such tool is CTC++ (Test Coverage Analyzer for C/C++). Some program languages include lots of different bug locating tools. There are for example Purify and Bounds Checker for C++ developing. Different kinds of emulators and simulators are used in embedded systems. /5 p. 278- 281/

Using of different analyzers and techniques make sure that proper piece of the software behavior is being observed. If testing is done inadequately it can have catastrophic consequences. For example, run time error lead to the destruction of the Ariane 5 rocket during its maiden flight. That might have been avoided with much thorough testing. /29/

Testing tools are divided into different groups depending on whether the program is executed during testing or not. Tools used to automate testing are normally divided into two groups. They are static and dynamic analysis tools. Testing management tools form a third group. Static analysis tools are used to find bugs in the software without executing the program at all. For example a compiler is considered as a static analysis tool when it verifies the code while compiling the code. Static analyzers are commonly used to more in depth analysis of the code. They can help with predicting the possible problems and troubles code can cause. For example tools like CMT++, QA C/ C++/ Fortran and PC-Metric are used in static analysis. /1/

Real time testing and analysis tools need proper environment in order to be useful. Normally environment is built using different test drivers. The drivers can be a part of some other program used in testing or own independent program. Testing itself is executed by inserting different outputs to testable program. Inserts can be done in the form of test scripts. Scripts can be implemented using for example Python or Ruby. Test scripts are easy to use repeatedly in the same test because when the script is done, only some modification might be needed. /1/

Dynamic testing can be done using some kind of simulator which simulates the parts which cannot be executed directly. For example, programs like CTC ++, QC/Coverage, C-Cover and Pure Coverage, can be used for quality and coverage analysis. Simulators like, SQA LoadTest- and PurePerformix can be used for simulated stress- and load testing analysis for multiple users. /1/

Because there are so many tools used for static and dynamic analysis, only couple of the best known tools is introduced using different program languages. For example, multilanguage programs are: Axivion Bauhaus Suite for programming languages Ada, C, C++, C#, and Java code. The program includes several features like architecture checking. /50/

Sonar is used to manage unit tests. It provides useful features like code complexity and code standards verifying. It also includes comments and potential bug problem identifying. It supports languages like, C, COBOL, and C#, Flex, Java, JavaScript, PHP, PL/SQL, Visual Basic 6, Web, XML and Python. /50/

The following tools are used with .NET. CodeIt.Right combines the best practices of static and dynamic analysis. CodeIt.Right can help correct automatically code and violations. C# and VB.NET are supported. JustCode is used as an add-on to Microsoft Visual Studio 2005/2008/2010. It is used for real-time, system-wide code analysis and it is supported by C#, VB.NET, ASP.NET, XAML, JavaScript, HTML, Razor, CSS and multi-language systems. /50/

C/C++ is used with Astrée and Eclipse. Astrée searches for run time errors and assertion violations. Eclipse software is an IDE (integrated development environment) which includes a static code analyzer /50/

AgileJ Structure Views and Jtest are used with Java. They both use reverse engineered Java class diagrams in static code analysis. Perl::Critic is a tool that uses the common practices with Perl. PerlTidy checks the syntax and act as a tester for Perl practices. Python uses tool called Pychecker for checking the source code and Pylint is used as the code analyzer. /50/

Selenium framework can be used with Python in web application testing. Selenium has a browser add-on, which is Selenium IDE. It is used for action recording and playbacks. A user can record browser interactions and then play them. /52/

Dynamic analysis tools are executing the program when analyzing it by either virtual machine or for real. Dynamic analysis needs enough inputs for it to be effective and for the analysis to produce interesting behavior. Dmalloc and Purify tools can be used for memory allocation and finding memory leaks. /52/

Many systems use functions to retrieve data from larger database and to display it when a user is requesting it. Usually changing the data brings problems that the interface has changed more than the requested data. Something is needed to tie the data and the interface together. Model-view-controller (Figure 9) is a design pattern that shows user and the model interactions. The model includes application data and business rules. According to the Wikipedia article about Model View Controller, “The central idea behind MVC is code re-usability and separation of concerns. “ /53; 54/

In the Figure 9, a controller receives a request from the user, to change to the editing state. The model notices changes and the view requests the information that is needed to update the output. The model is managing the data and the application behavior. It reacts to state changes and the information coming from the controller and the view. The view takes care of the information display. The controller is saving the user inputs and informing the model or the view for changes. /53; 54/.

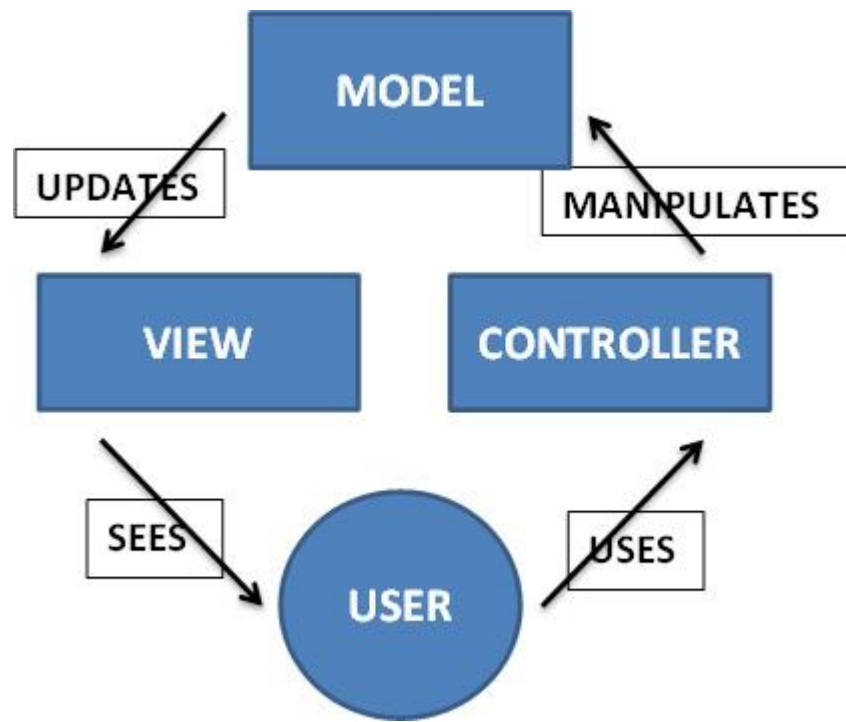


FIGURE 9: Model View Controller. /53/

The model view controller helps in testing process. If there are many components, the testing will become difficult and the components often require much more complicated scenarios. The model view controller is keeping all the actions separate making the components easier to test individually. Using the model does not mean that there is no need for user interface testing. The model usage is reducing test cases when there is a difference between the model and the presentation logic. This allows the model to be tested independently. /54/

The following test automation tools are mostly used: Labview, IBM Rational Functional Tester, Selenium, Visual Studio Test Professionals, Abbot Framework, Atlassian Jira and AutoIt.

Labview (Laboratory Virtual Instrumentation Engineering Workbench) is National Instrument's system design platform and development environment. It uses dataflow programming language. Labview's advantage to the other systems is that it includes many different types of instruments. More often used items are preinstalled which saves development's time. Labview requires very little coding experience which makes the programming threshold very low.

IBM Rational Functional Tester is used in automated testing to copy human behaviour and actions. It is normally used in automated regression testing. It records user actions

of the application under testing and allows testers to create test scripts. The scripts are made in either Java or Visual Basic.net.

Selenium is a web application testing framework. It provides recording tools for mouse or user action recordings to create tests without learning any specific scripting language. User can create tests using PHP, Python or Ruby for example. Selenium can be used on Windows, Linux and Macintosh platforms.

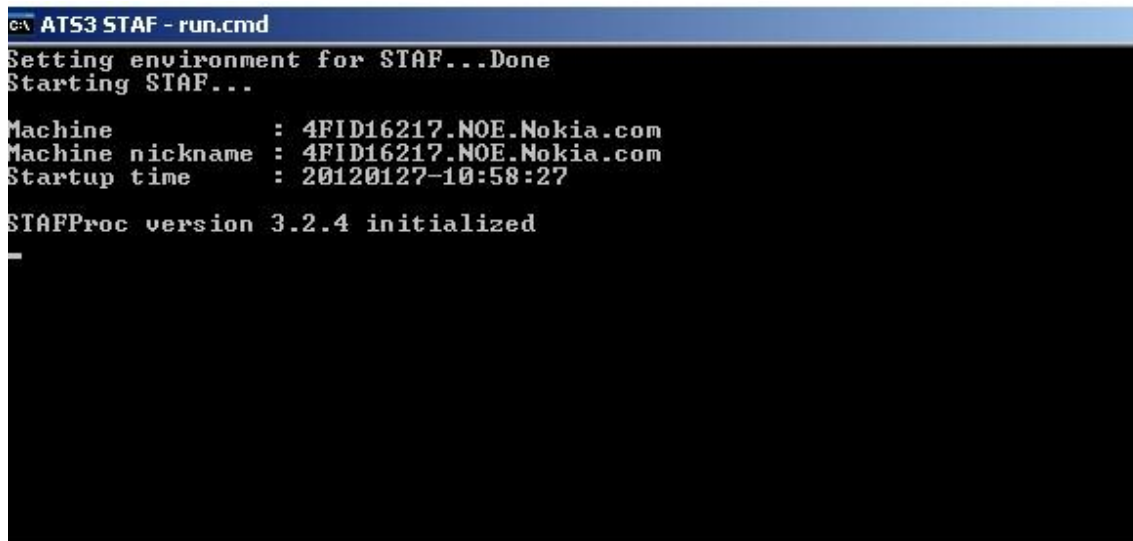
Visual Studio Test Professionals is Microsoft designed testing tool to help testing workflow between testers and developers.

Abbot framework is for automating Java graphical user interface components and programs. Atlassian Jira is project / bug management application.

AutoIt is a scripting language designed for Windows GUI automation. /28; 36; 55; 56; 57; 58/

4. ATS – AUTOMATED TESTING SYSTEM

ATS is automated testing system which is used to automate software testing. The system consists of one server and at least one client (mobile device), which is harnessed to a worker PC (see Figure 10). The worker window shows the name of the worker PC and the time when the worker was started. All the errors with the worker are also displayed in this window. The server can also act as a worker PC and a mobile device can be harnessed to the server. Worker is a kind of testing module with ATS application installed. The workers (Figure 10) communicate with the server and the database and execute the defined tests. Each of the PC's, which has the worker installed, is part of the ATS network, and each of the workers has STAF included. The Software Testing Automation Framework, STAF is an open source test framework. It will provide the basic network services used in the ATS. /1/



```

C:\ AT53 STAF - run.cmd
Setting environment for STAF...Done
Starting STAF...

Machine       : 4FID16217.NOE.Nokia.com
Machine nickname : 4FID16217.NOE.Nokia.com
Startup time  : 20120127-10:58:27

STAFProc version 3.2.4 initialized

```

FIGURE 10: The worker that is active

ATS server is Java based application which “supervises” test case executions. Tests are divided among the workers and the results are recorded into the database. ATS-system can flash the software image to the mobile device before testing. The test system collects the correct test sets which are stored in the phone in “Software B”. The architecture which is used in the ATS system is shown in the Figure 11. /1/

One PC is working as a server which takes care of the test runs. The device under testing can be located anywhere as long as it is harnessed to the server. The working system requires that there is MySQL database, Java SDK and Tomcat Apache Web server. The worker must be installed to every PC where test runs are executed. In order to execute the tests from windows command prompt Perl package must be installed to the worker PC. After the worker is installed then the ATS-system can be installed to PC. During installation, all the necessary items are defined, like TCP-ports, database used, server, user names and passwords. Testing starts after the device under testing has been registered to the client. Register file or properties file (See Table 1) is used to define the name of the device, device type and the connection type. /1/

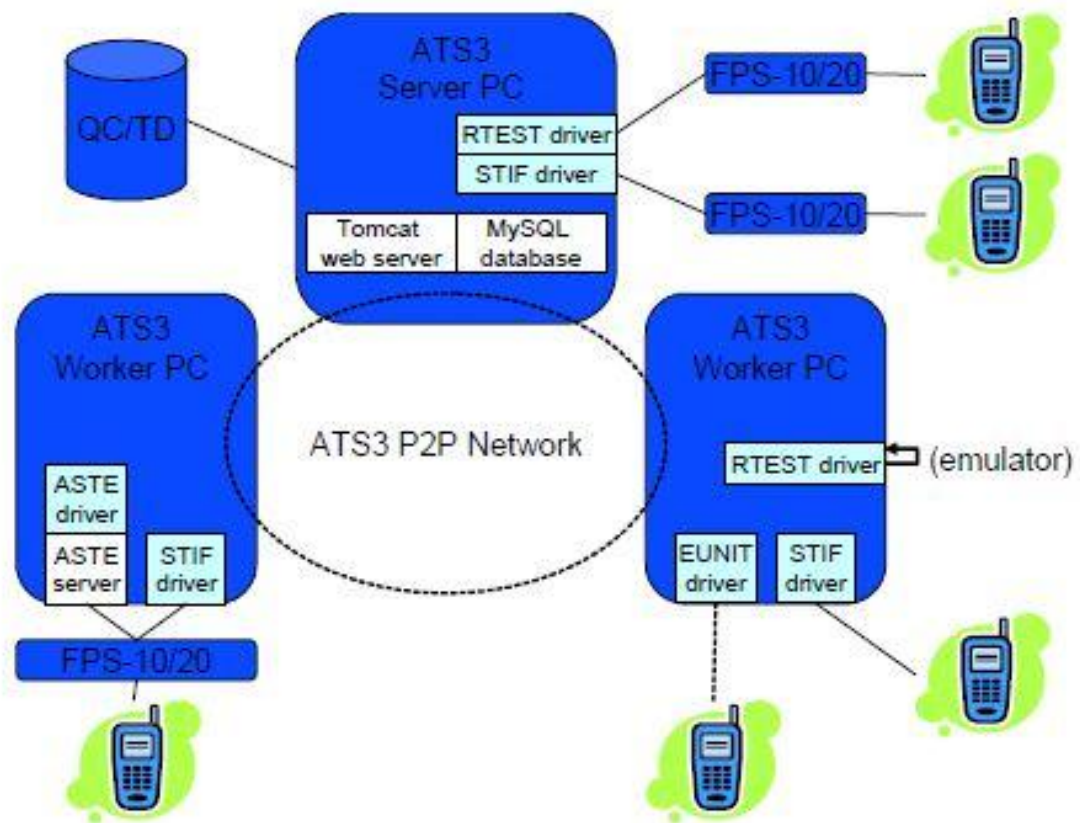


FIGURE 11: Automated testing system architecture /1/

After the device has been registered to ATS-system, test plan must be defined. The system needs to know the testing data, test sets, device under testing and testing framework. When the plan has been created test run can be started. An old plan and/or imported plan can also be used for testing. For completely automated testing, the plan needs information when the tests are executed, where they are executed and how many times they are executed. Since this system has been in use for many years, there was no need to create a new test plan. /1/

TABLE 1: An example of the registered properties file /1/

```

NAME=Test phone1
CATEGORY=hardware
TYPE=Test phone1
CONNECTION=HTI
CLASS=StifTestableDevice
HARNESS=STIF
# reinstall files after reboot
REINSTALL=false
BUILD=urel
PLATFORM=armv5
#FLASHER=FPS10Tool
#single (.img) multi (.img.mcu && .img.ape) or optional_multi (.img.mcu (if exists) && .img.ape)
IMAGE_TYPE=optional_multi
#FPS10Tool or HTI
#REBOOTER=FPS10Tool
BOOT_TIME=5
#DataGateway port
DGW_PORT=
#Initialisation of DataGateway connection (note: only one should be uncommented)
#USB (DKU-2) Connection
#DGW_INIT_STRING=BUS_NAME: USB
#Standard FBUS Connection DAU-9(x)
DGW_INIT_STRING=BUS_NAME: FBUS PORT_NUM:1
#ComBox FBUS connection
#DGW_INIT_STRING=BUS_NAME: COMBOX PORT_NUM:1 COMBOX_DEF_MEDIA: FBUS
#ComBox Direct FBUS connection
#DGW_INIT_STRING=BUS_NAME: COMBOX PORT_NUM:1 COMBOX_DEF_MEDIA: FBUS BOX_TYPE: NEW
PROTOCOL: FBUS
#FPS-10 TCP-IP Connection ACTIVE_MEDIA set to FBUS, could be USB also
#DGW_INIT_STRING=
#FPS-10 USB Connection, ACTIVE_MEDIA set to FBUS, could be USB also
#DGW_INIT_STRING=
#FBUS Connection through USB to Serial converter DKU-5, CA-42, DKU-8
#DGW_INIT_STRING= BUS_NAME: DKU-5_FBUS PORT_NUM:80
#IrDA connection
#DGW_INIT_STRING=BUS_NAME: IRDA

```

ATS is using a testing framework called STIF. It is a framework for Symbian user interface components. It is used for test case implementation and for testing itself. STIF is using different testing interfaces, like STIF Console UI, STIF Series 60 UI and STIF ATS Interface. STIF includes error exception handling and memory consumption observing. It can also execute multiple tests at the same time. The mobile device has STIF and HTI installed /1/

HTI is testing interface for Symbian based mobile phones. Harmonized Test Interface (HTI) can be used to control test target. PC side test services are included in HTI which has corresponding services on Symbian side. Communication mechanism is also

included in HTI framework. HTI is used with the other testing tools as it is included in many RnD tool packages. /40/

Test system has a web based user interface (Figure 12) where test executions can be followed in real time. Users have possibility to end test run and restart it. Real tests are executed in the mobile device under testing. Figure 12 shows ATS3 test execution screen, where test runs are followed. Left side of the figure shows test sets that have been fetched from the Quality Center testing tool. In reality there are much more test sets but this example shows only four sets. Test set names have been blackened due to the confidentiality issues. Name of the test sets come from the Quality Center. Test plan shows the name of the worker computer which was used to fetch test sets. Test status shows tests running if testing is not finished. After execution is done, the status changes to finished. Progress bar shows how the execution is proceeding. If all test cases are passed the bar is green and if there are errors the bar is red. Right side of the Figure shows execution start-and end times /1/

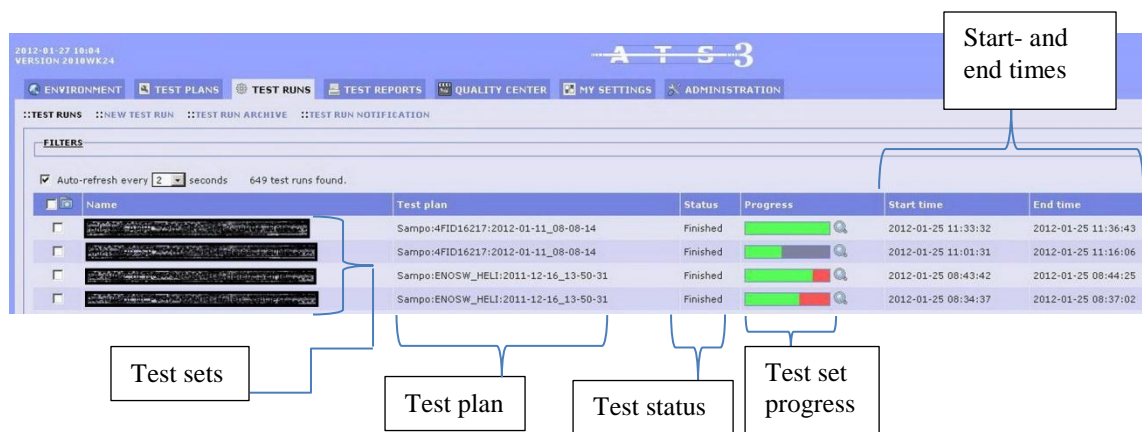


FIGURE 12: ATS3 web interface

Mobile device can be attached to PC which has worker installed via USB or FPS-10 / 20. Web interface in Figure 13 shows the test sets executed and their results. The result view shows which test cases have passed and which have failed. Green means that everything is ok and red means that something is wrong, but not necessarily failed. In Figure 13 all the tests are passed. Tester can see the individual test cases and their results by clicking the appropriate test case name. In this view tester can also see test logs of individual test cases. /1/

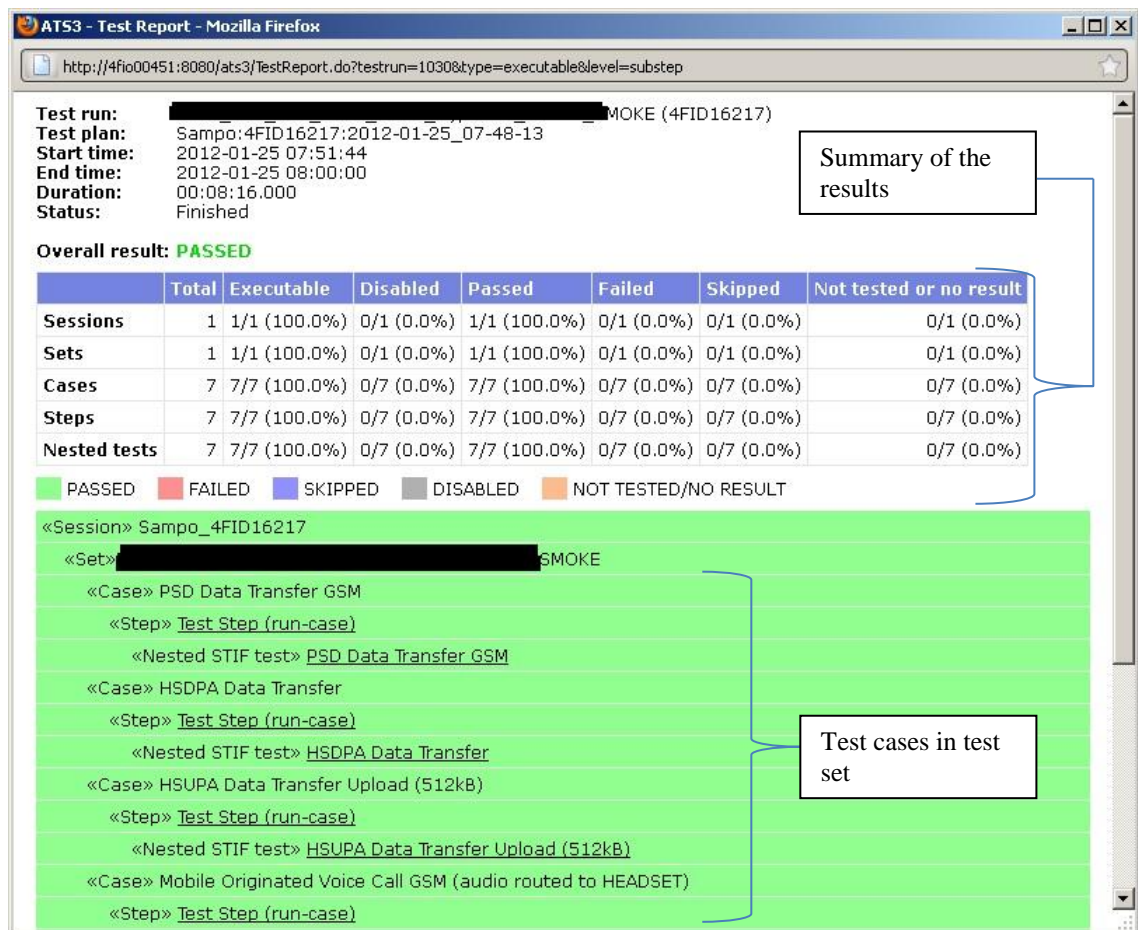


FIGURE 13: An example of the result view

ASTE or Automatic System Test Engine is a user interface for UI testing. UI is tested to make sure that the interface looks appropriate and clear without errors. The most important components are keywords and Test Execution Management Tools. Keywords describe actions that can be done with the device under testing. /1/

5. SYMBIAN PHONES

Symbian OS was Nokia's operating system for mobile devices and smart phones. It was derived from Psion's EPOC. Nokia purchased Symbian in 2008 and formed an independent foundation called Symbian Foundation. Symbian OS and its interfaces, S60, UIQ (formerly known as User Interface Quartz) and MOAP(S) (Mobile Oriented Applications Platform), were given to the foundation by their owners in order to

create an open source Symbian Platform. This platform was supposed to be a successor to the Symbian OS. Symbian was made open source in 2010. /59/

The first step for Symbian was called EPOC (EPOC interface in Figure 13). The name EPOC meant “epoch”, the beginning of an era. The engineers changed it to “Electronic Piece of Cheese”. EPOC was followed by versions EPOC16 and EPOC32 (Releases 1 to 5). EPOC16 version was developed in the late 1980s and the early 1990. It was using a single processor and the languages were assembler and C. It supported single user's pre-emptive multitasking and had a keyboard operated graphical interface. The first ever EPOC systems were delivered in 1989. Later name was changed to EPOC16 to separate it from the newer version. /59/



FIGURE 14: EPOC interface /59/

EPOC32 was again referred to as EPOC in the 1997. Later it was changed to Symbian OS. There was lots of confusion because the name changed constantly to distinguish the older version from the new one. EPOC16 and EPOC32 were completely different operating systems in spite of the similar names. EPOC32 was written in C++ with completely new codebase. It had EPOC16's multitasking and also a memory protection. The developers made separate interface to the OS. In 1998 the mobile device

manufacturer Psion Software was changed to Symbian Ltd. It was a big venture between Ericsson, Motorola and Nokia. When Epoc 32 Release 6 became available it was known as Symbian OS. /59/

Nokia acquired Symbian Ltd in 2008 and became the major contributor for Symbian. Nokia now had all the resources for Symbian core and the user interface development. Due to the lack of the funds, the foundation became a licensing only foundation. Accordingly Nokia announced to take over the platform management. /60/

The new Symbian OS was the base of the modern smartphones. Nokia's communicator was the first open Symbian OS phone in 2001 and it had Bluetooth support. Symbian era had started with almost 500 000 Symbian phones shipped in 2001, exceeding 2.1 million in 2002. The UI's continued to evolve. Ericsson's "Ronneby" design was merged with DFRDs or Device Family Reference Designs and it reached the markets as Nokia Series 80 UI. It evolved to Nokia Series 60 UI. It was the UI of the first true smartphone. Nokia 7650 was featuring Symbian OS 6.1 and it had the first built in camera, with VGA. /59/

Symbian OS 7.0 in the 2003 was the important release with modern user interface. It added MMC card, EDGE and Ipv6 among other things. By the end of the 2003, Symbian phones were selling almost one million per month. As Symbian gained popularity, security became an issue. The first mobile worm for Symbian OS was developed. It was using Bluetooth for spreading to nearby phones. /59/

Symbian OS 8.0 would have had an advantage of the choice of the two kernels, EKA1 and EKA2. They look similar to user but behave differently on the inside. EKA1 had the compatibility of the old device drivers and EKA2 was a real time kernel. Version 8.0 included a support for CDMA, 3G, DVB-H and OpenGL. Symbian OS 8.2 was an improvement over the previous one, with both kernels. There was no added security layer. The first smartphone with OS 8.1 installed was Nokia N90 and it was the first ever Nokia Nseries phone. Symbian 9.0 was only for internal use and it was the final OS with EKA1 version. At this point Symbian needed changes to its security because the viruses started to spread among Symbian phones. In 2005 Symbian OS 9.1 added security features to the OS, including a security module to help with mandatory code signing. SIS files used in the mobile phone must be signed in order for them to get

access to certain APIs. (System capabilities, restricted capabilities and device manufacturer capabilities) Now all the applications can be signed for free. Also a Bluetooth 2.0 support was added to this version. /59; 60/

Version 9.2 brought a support for OMA Device Management. Nokia introduced phones E71, E90 and N95. The next version included some improved memory management, Wifi and HSDPA. In 2007 Symbian version 9.4 was announced. It promised even 70% faster application launching. It was the basis of the Symbian^1, better known as the S60 5th edition. Symbian^2 was aimed only for Japanese markets and it is not used by Nokia. Symbian^3 or Symbian OS 9.5 version was a really big improvement for Symbian. It used hardware-accelerated graphics and had a new user interface. It also had portrait QWERTY keyboard and a new browser. Symbian^3 and the previous versions used native WebKit browser. Earlier versions used Opera mobile. Symbian was the very first mobile device to use WebKit browser. Updates were based on Qt framework. After that, Symbian^4 was supposed to be the next major release. But it was discontinued by Nokia. Symbian^3 update was named Symbian Anna and the following big update was called Symbian Belle (Figure 15). In 2012 the update was changed to Nokia Belle (Figure 15). It was bringing lots of very welcomed improvements to Symbian. They were pull-down bar to home screen, much deeper NFC integration and resizable home screen widgets. It also added the amount of home screens from three to six. Nokia Belle has currently the support for 48 different languages. Some people say that the update was too little and too late for Symbian. /59; 60/



FIGURE 15: Nokia Belle /60/

In 2011 Nokia announced to start using Windows Phone in smart phones. The number of Symbian devices started to reduce dramatically. The new competitors gained a lot of market share. During 2009-2011 the big Symbian supporters Nokia, Motorola, Samsung, LG and Sony Ericsson decided to give up Symbian. Nokia outsourced Symbian development and maintenance to Accenture. /60/

Nokia 808 PureView remained Nokia's last Symbian phone which is introduced in the Figure 16. It is Belle powered Symbian based smartphone featuring PureView technology (see Figure 16).

PureView uses a pixel oversampling, which means a combination of many image pixels in one image. The phone is then able to reduce the image to lower resolution. The phone uses Symbian Belle Feature Pack 1, upgradeable to Feature Pack 2 with 1.3 GHz ARM11 CPU. It features high resolution f/2.4 lens and 41 megapixel 1/1.2" sensor, which was the largest and the highest sensor used in mobile phone. The image sensor has its own image processor chip which reduces the external processing needs. This allows capturing to high resolution images. The camera uses digital zoom but it retains high resolution because of the large 41 megapixel sensor. Autofocus is continuous in all shooting modes. The phone also features Bluetooth 3.0 and secure NFC. /60/



FIGURE 16: Nokia 808 PureView /69/

Symbian used C++ programming which was really hard to learn from the beginning. C++ required really special techniques that it made even really simple programs harder to implement. These issues are no longer valid when C++ is used with the Qt SDK. /60/

According Nokia developer Wiki, The key features of Symbian OS are:

- Good performance with minimal power and memory demands
- Ability to run different applications simultaneously (multitasking)
- Use of agreed-upon technical standards for robustness, portability and interoperability
- Object-oriented architecture
- Optimization of memory management
- Minimization of runtime memory requirements
- Securing the communications and data storage
- Support of internationalizing and localization with Unicode character sets
- Diverse API library. /64/

Symbian kernel called EKA2 is supporting a fast single core phone which executes user applications and protocol stack. Kernel (Appendix 1, Figure 17) contains device drivers with networking, memory management and telephony support services. Symbian System Kernel is located in the base of the OS layer as seen in the Figure 18. Symbian is using microkernel which means that kernel is managing the whole system resources and the memory. It is separating the user interface and the engine. It takes care of the dividing time between the different applications and the tasks needed by the system. Higher levels of the layer are providing communication services like TCP/IP, IMAP4, and SMS. Symbian components are supplying different data management methods and security. /60; 64/

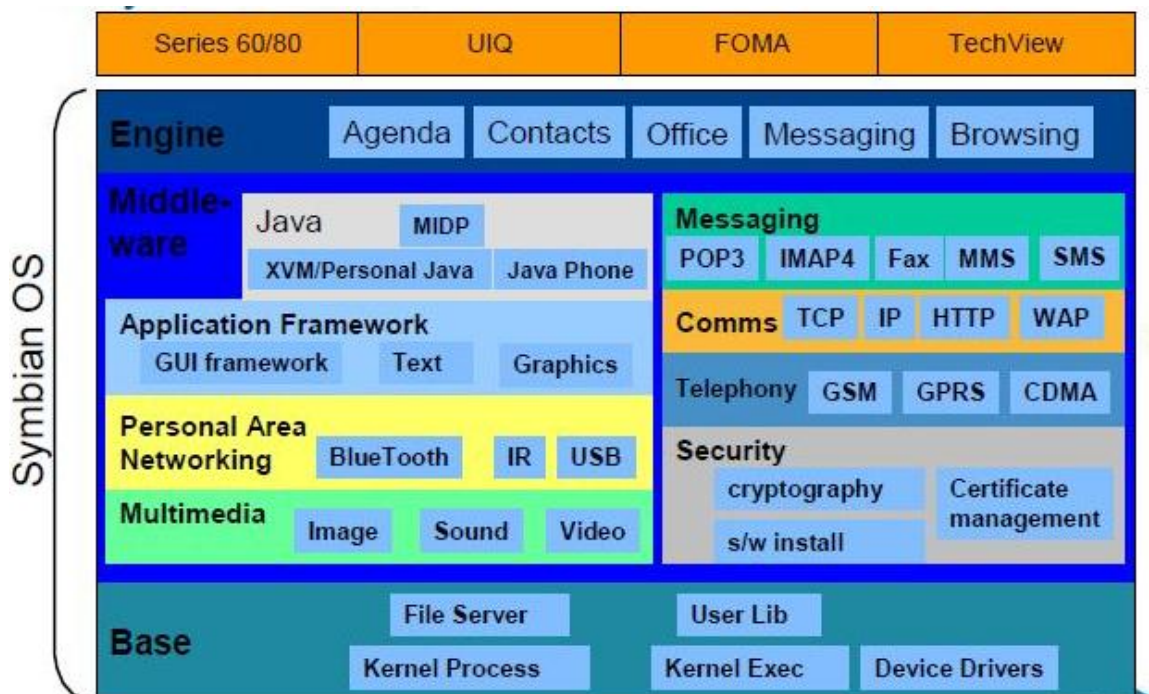


FIGURE 18: Symbian OS layer architecture /63/

According to Symbian Wikipedia article, “Symbian contains the following layers (See Figure 18):

1. UI Framework Layer
2. Application Services Layer
 - Java ME
3. OS Services Layer
 - generic OS services
 - communications services
 - multimedia and graphics services

- connectivity services
4. Base Services Layer
 5. Kernel Services & Hardware Interface Layer.” /60/

Symbian base layer is the lowest layer where user side operations can reach. It includes plugins, file server and central repository. It also has the text window server and the text shell. Those two services are the basic services which can form a functional port without higher layers. /60/

Symbian is widely spread operating system and is also a target to viruses. Usually viruses find their way into the phone by Bluetooth. At this point, none of the viruses have taken advantage of the Symbian flaws. Viruses are usually asking a user to install a program. At this point, the operating system warns that the program can't be trusted. It is up to the user to quit the installation which may harm the operating system. /60/

As explained before, Symbian 9.0 started using the much tighter security model. It used permissions per process, not per object. Installed software is unable to do any monetary damage without being digitally signed and therefore it is traceable. The developers, who can afford to have the applications Symbian signed, can apply that via the Symbian Signed program. They also have the option of self-signing the applications. However, some of the operators have disabled certificates which are not Symbian signed. /60/

Most of the viruses or hostile programs found in Symbian operating system require user input to run. Drever.A is a mean Trojan that attempts to disable antivirus software and Simworks automatic startup. Locknut.B spreads through SIS files, and it pretends to be Symbian S60 patch file. After installation it causes critical system component to crash and therefore prevents any application launches. Mabir.A is basically same as Cabir. They both spread through Bluetooth functionality. When the virus is activated, it will go through Bluetooth search and sends copies of itself to the phone it has found. Fontal.A installs the file that causes the phone to fail during reboot. If the phone is used while infected, it will stay on reset loop and cannot be used without resetting all the data in the phone. Luckily, Trojans cannot spread by themselves. User must give permission to the file to run or get the file from untrusted sources.

The platform security can be removed in version OS 9.1 and onwards, allowing users to have access and to alter the previously locked areas of the operating system. This makes

it possible to run unsigned code in the phone. New threat for Symbian was introduced in 2010. A hacker found a backdoor for S60 devices. The hacker modified the older version of the operating system and integrated the backdoor as a reverse shell. The smartphone could then be remotely controlled. Luckily, the attacker would have to first have an access to the phone and connect it to a computer. After the installation, the backdoor is revealing the phone IP address. The attacker has then the ability to read the emails, text messages and record phone calls. The only way to remove the backdoor is to reflash the phone firmware. /60; 65/

Symbian platform's main security goals are:

1. Phone's integrity needs to be protected.
2. Users need privacy: The access from outside needs to be limited.
3. Controlling accesses to networks to prevent high costs: Unauthorized data usage can be expensive. /66/

The security limitations are:

1. Developers need to be able to use it easily.
2. Security cannot cause any slowness to the system.
3. Security policy is shown to the users when they understand it. Otherwise it needs to remain invisible. /66/

Symbian security is based on three things:

1. Symbian operating system process is the smallest entity which can be granted permissions. In other words, the unit of trust.
2. Access to sensitive resources is controlled by the capabilities.
3. Data caging is used for protecting the files for unauthorized accessing. /66/

The unit of trust is a smallest entity, a process which can be given to set of permissions. The phone has only one user which means that the security is not the same thing than in desktop computers. In phone the security is about controlling the applications, and whether they can be allowed to run. In Figure 19 the trusted computing base is maintaining the integrity of the device and for applying the fundamental rules of the security. The parts of the trusted computing base, TCB have unrestricted access to device's resources. That is why all the code involving TCB is reviewed very thoroughly. The trusted computing environment, TCE is outside of the core. TCE consists of the

Symbian key components that protect the system from misuse. The final level of trust is called “Applications”. It is the most outside level in the model. This does not usually have any capabilities that can hurt the system or endanger the integrity. The applications use other resources to access the sensitive components. /66/

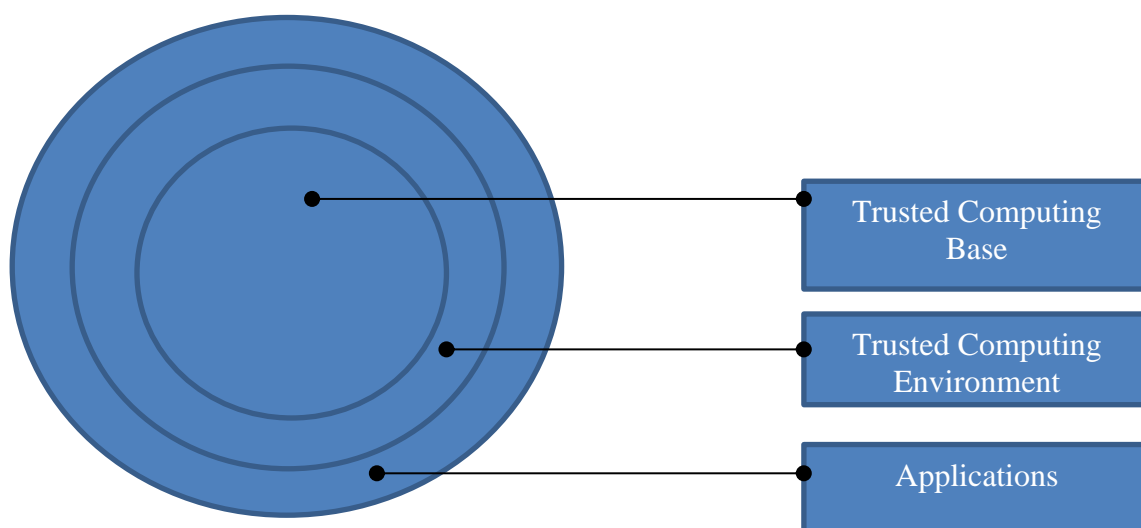


FIGURE 19: The levels of trust /66/

To make sure that the owner is trusted, Symbian uses a capability token. In verification of the capabilities, there are some rules for understanding the security:

According to the capability model rule 1, every process has a set of capabilities and they are valid during the process lifetime. This means that the Kernel stores the capability of each of the process in the memory which can't be accessed by any user mode processes, in order to prevent tampering. /66/

According to the rule 2, the process cannot load a DLL (Dynamic Link Library) that has the smaller set of capabilities. /66/

Rule 2b is an addition to the rule 2: When DLL has a smaller set of capabilities; it cannot be statically linked to DLL.

The used DLL code must run at the same capability level as the loading process. The DLL capabilities are different from the process capabilities; they are policies by the loader. This means that every DLL is only verified at the load time. /66/

Data caging is another key concept of the platform security. Data caging means file access control. The idea is not to control who has the access to the files but which process has the access to the certain file. /66/

According to the rule 3, the access rules of a file are determined by its directory path. Only the most trusted processes have the access to \sys directory. Only the TCB processes can write to \resource directory. It contains files that are not expected to change. Private directory can only be accessed by TCB process, appropriate process or backup server. Any other process has no rights to either read or write to that directory. All the other directories are completely public space. /66/

Each program gets a unique identifier, called SID (the secure identifier). Kernel can give a private space for each of the programs that are requiring it. Each SID gets assigned when compiling and they are associated uniquely with one executable. If the SID is not defined, UID3 is used as SID. If there is no UID3, then the program gets a value “KnullUid”. A Vendor ID (VID) is another form of UID. It is a software identifier for vendors. It is used as runtime mechanism to check that the binary is coming from known source. VID can be defined by specifying a Vendor ID in the program’s properties file. Most of the developers are not using VID and the value is then zero as it is when the ID is not found. Like the SID, the process VID will be the same than the executable. VID can be used to restrict the use of some application or the network for the certain vendor. /66; 67; 68/

How the capabilities are controlled is summarized in the table 2. TCB gives permissions to executable and read only resources. “AllFiles” gives reading rights to the entire file system. It grants read/write access to private directories of the other processes. When looking at the table 2, we notice that TCB files under \sys need “AllFiles” to read. OwnSID under the private directory is reserved for application data. “Other” directory is public space. Capabilities should not be each other’s subsets and they should not create any ambiguity about the access rights. “AllFiles” creates an access to the entire system without giving write access if not necessary. Description of what is the capability accesses used for can be seen in the table 2. /66; 67; 68/

TABLE 2: The access rules /66; 68/

	The capability required to		Used for:
	Read	Write	
\resource	none	TCB	Read-access for applications
\sys	AllFiles	TCB	Applications can only be executed from here.
\private\<ownSID>	none	none	Application data
\private\<other>	AllFiles	AllFiles	Required to access the other applications' data
\<other>	none	none	Unrestricted data

All the executable files are stored under the \sys\bin. File server's duty is to make sure that only TCB processes are able to write into that directory. This ensures that the executable is always signed and verified. Otherwise the executable is not loaded. Because every executable file is verified during installation, any break in the rules will cancel the installation. This is an effective way to protect against Trojan horses. /66/

Symbian operating system uses Symbian's relational database and the central repository for file sharing. But when there is a need to share file between the processes it is done under controlled conditions. If there is a message with an attachment, the viewer needs to be opened without revealing all other possible attachments to the viewer. The file is opened in the mode that cannot be changed by the receiving process without the file server rechecking the access policy against the file credentials. The receiving process does not get the access to the parent directory of the receiving file. The access is granted only to the shared session and the file handles. When sharing files, the session should be opened only for the specific file sharing. If the session is not closed properly after use, any other files opened by the process are accessible to the receiving process. This makes it possible to increment the handle numbers and to gain access to the other files. Ignoring this rule causes a big security hole in Symbian. Figure 20 summarizes the whole security concept explained in this chapter. /66/

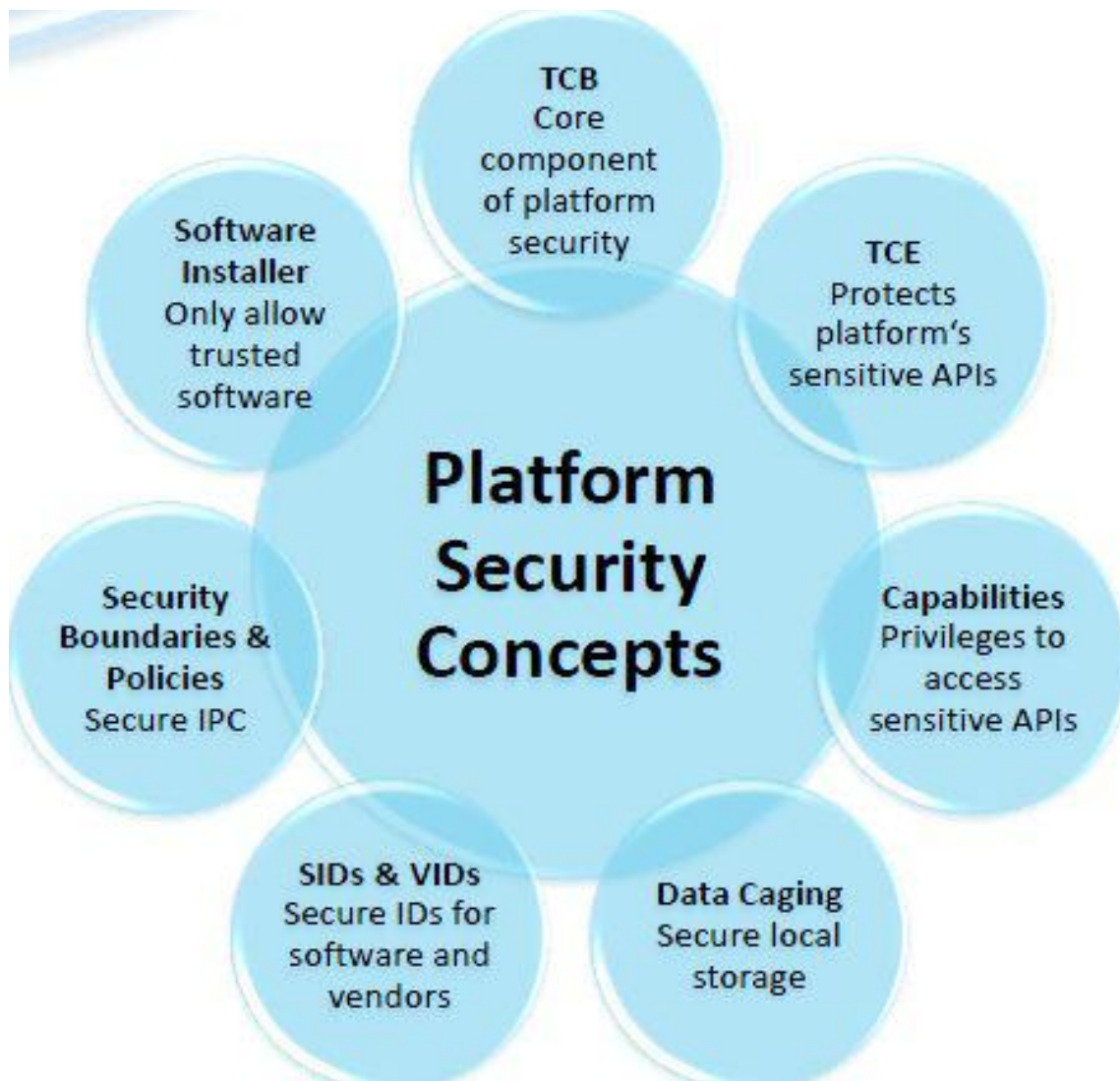


FIGURE 20: Security concepts /68/

6. MeeGo

MeeGo is a free Linux based mobile device operating system. It can be used in notebooks (interface in Figure 21), tablets, computers, TV's and in some other embedded systems. It is currently hosted by the Linux Foundation. The future “birth” of MeeGo was announced in 2010 and it started as a joint project between Nokia and Intel. Their aim was to merge Intel's own Moblin and Nokia's Maemo. Since the collaboration, Nokia decided to keep the middleware components and the packaging managers of the Harmattan system. The device is fully compatible with MeeGo and the end distinction is only minimal. /4/



FIGURE 21: MeeGo netbook Screen shot example /4/

Harmattan was originally going to be Maemo 6. Only the UX (user experience) name was changed to MeeGo / Harmattan. In 2011 the first device was published by Nokia, it was Nokia N9. /4/

The previous versions are still referred to as Maemo. MeeGo was going to be Nokia's future operating system. However in 2011 Nokia announced that their new operating system will be Windows. After that MeeGo became Nokia's research subject. Later that year, Intel announced a new mobile platform called Tizen. It started as joint project together with Samsung and Intel. /3; 4/

Now when Intel too had given up MeeGo, the community stays still strong. The developers from the project “Mer” are continuing the MeeGo without Intel and Nokia. In 2012 a Finnish company, Jolla announced the ”rebirth” of MeeGo as they will continue with “Mer” /4/

The Mer project was a reimplementations of Maemo and it was abandoned during the start of MeeGo. After Intel and Nokia gave up MeeGo, the Mer project was again

revived and rebased on MeeGo tools. It is now an open community project which provides only the core, with capability of running various UX's (user experiences). Mer is considered to be a successor of MeeGo. It was initially announced to be a continuation of the MeeGo project. However, there is not much shared architecture between the projects. The continuation of MeeGo is now called Tizen. It does not use Qt framework which was the core of the MeeGo's API (application programming interface). That is why Tizen is not technically a “spin off” of MeeGo. /4/

MeeGo is intended to run on different platforms as explained in the beginning. The MeeGo core is the same in every platform. Only the user experience (“UX”) layers are different. There are several graphical user interfaces inside MeeGo, and they are called “User Experiences”. The netbook in the Figure 21 is a continuation of Intel's Moblin. It uses Linux applications integrated in the GUI. The Handset UX is based on Qt but it will include compatibility for Moblin. MeeGo has the support for ARM and Intel processors with enabled SSSE3 and btrfs file system (B-tree file system). /4/



Since N9 (Figure 22) was left to be Nokia's only MeeGo device, it is introduced here just as comparison to Symbian operating system. The operating system is not exactly MeeGo, but it is referred to as MeeGo instance. N9, a code named “lankku” was a successor of Nokia's Maemo device. N9 was expected to be launched at the end of 2010, approximately one year after N900. Nokia N9 included 3 processor units with OpenGL supported GPU. It could process 14 million polygons / second. A digital signal processor takes care of the image processing for camera, audio, telephony and data-connections. The screen is almost 4 inches wide

FIGURE 22: Nokia N9 /39/ capacitive touchscreen with resolution of 854×480 pixels. N9 uses scratch resistant gorilla glass with coated screen to help the visibility of the screen in the sunlight. Proximity sensor deactivates the display when a face is near the screen. That is a mandatory in order to prevent accidental screen presses, for example during a call. Accelerometer rotates the screen to portrait/landscape mode when the phone is rotated. Camera in N9 has dual LED flash 4xdigital zoom and the sensor size is 8.7 megapixels. The phone is also supporting NFC technology (Near Field

Communication). It can be used for sharing contacts, images or music with other NFC devices. The real idea behind N9 is the Swipe experience. The horizontal swipe gesture navigates between the three home panes. /39/

Users' reaction to N9 was positive all around the world, especially because it was an open source platform where the community could create applications for it. In addition most of the applications are free. It has polycarbonate design with NFC capabilities. It was told to be "the most beautiful phone ever made". The phone's only minus in reviews came because of Nokia's decision to drop the MeeGo for Windows. /39/

The Core of MeeGo uses a Linux distribution, parts of Nokia's Debian based Maemo and Intel's Fedora based Moblin. It is among the first ones to use the btrfs file system (B-tree file system) and RPM repositories (RPM Package Manager) as default. MeeGo applications are created with Qt framework and Qt Creator. OpenSUSE's build service is used for compiling. /4/

MeeGo architecture is divided into three parts: Layer view which shows the separation of the layers and UX's (user experience). Domain view shows the grouping of domains and API (application programming interface) view shows the grouping of different API's. /38/

The Figure 23 shows three layers of MeeGo, user Experience, Application API, and Core OS layers. The UX layer shows multiple platforms. This Figure shows the support situation for MeeGo 1.1. Application framework is provided for each of the device profiles by the UX layer. Different user experiences use different frameworks.

MeeGo API is included in the Application API layer. Interface for the development is included in API layer. In Core OS all the middleware/OS service domains and the hardware adaptation services are included. Kernel is in the Core OS layer. /38/

Core OS is grouped in the domains according to the functionality. The Figure 23 shows all the domains according to the MeeGo security architecture:

1. Security domain includes the framework for security and enablers.
2. Data Management has meta data storage.
3. Software Management domain includes package management and software lifecycle.

4. System domain includes management for device state and resource policy and sensor and context.
5. Location Framework is in location domain.
6. Graphics domain includes X11 (The X Window System), OpenGL (Open Graphics Library), input and Display drivers.
7. Essentials domain includes all the essential system libraries.
8. Multimedia related enablers and drivers are in multimedia domain.
9. Calendar, Contacts, Backup, and Sync are located in PIM-domain (Personal Information Management).
10. All the communication related items are in communications domain. They include VOIP, IM, Presence, Cellular Telephony, and IP Connectivity
11. Qt domain includes Qt, QtWRT and Qt Mobility.
12. Linux Kernel and core drivers are located in the Kernel. /38/

All the different platforms require platform specific hardware adaptations. Chipset's adaptation software is required to implement these changes. /38/

Hardware adaptation layer is located at the bottom of the figure 23. According to the MeeGo security layer architecture: Hardware adaptation is divided into the following subsystems:

1. Security
2. Sensor
3. Device Mode
4. Haptics and Vibra
5. Audio
6. Camera
7. Imaging and Video
8. Location
9. Cellular
10. Connectivity
11. Input
12. Display and Graphics. /38/

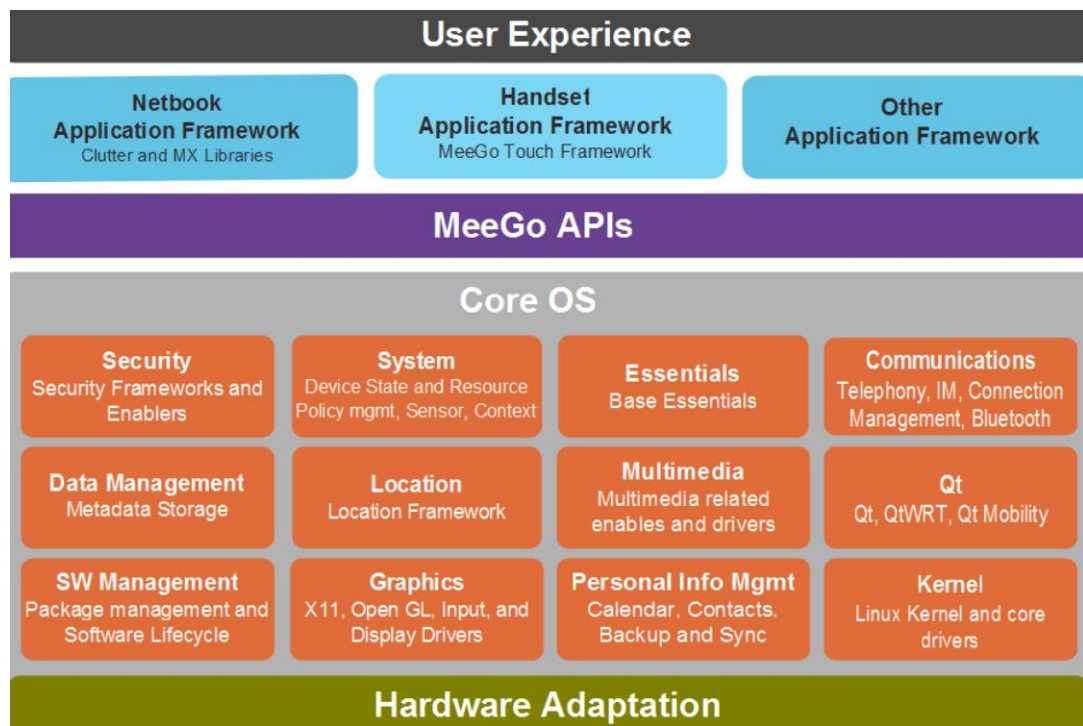


FIGURE 23: MeeGo architecture - Layer view /38/

The domain view is shown in the Figure 24 in appendix 2. Each of the twelve domains is divided into details according to the domain functionality.

Security provides security services and user identity for the system. The domain includes the following subsystems according to the MeeGo security architecture:

1. Accounts provide storage for the accounts.
2. Single Sign On provides secured storage for credentials, authentication framework and plugins for different services to use.
3. Integrity Protection Framework provides framework to protect executable integrity and configuration.
4. Certificate Manager stores security certificates.
5. Software Distribution Security takes care of the new application installations and update security aspects.
6. Access Control Framework serves devices access control policy.
7. Security Adaptation provides security and crypto services for the device. /38/

Data Management's subsystem is content framework, which has indexing, extraction and search capabilities.

Software management subsystem includes Package Manager. Software management offers functionality for packet manager and its functionality. It uses management tools to make software installation and updates easier.

System domain's responsibilities are managing the device states and mode changes, time management, all the device start up services, different sensors and policy control. System domain has the following sub domains which can be seen in the Figure 24 in appendix 2:

1. System Control is responsible for device state and time management.
2. Resource Policy has framework for audio, video, and system policy management.
3. Startup Services is responsible for all the components used in startup.
4. Context Framework provides high level APIs.
5. Sensor Framework offers interface to hardware sensors.
6. Sensor Adaptation provides specific plugins for sensor framework.
7. Device Mode Adaptation is an abstraction layer for device mode.
8. Haptics and Vibra Adaptation is also hardware abstraction layer. It is for Vibra and haptics devices. /38/

Location domain provides different location services. Figure 24 in appendix 2 shows that it has the following sub domains:

1. Location Framework provides location data combination of different sources.
2. Location Adaptation is hardware abstraction layer for location source devices. /38/

Kernel includes Linux kernel and device drivers. Personal Information Management domain provides user data management. It includes services for calendar, contacts and email. Domain includes the following subsystems (Figure 24 in appendix 2):

1. Interface for accessing calendar data is provided by Calendar Engine.
2. Interface for accessing contact data is provided by Contacts Engine.
3. Email data interface is provided by Email Engine.
4. Backup Framework offers backup services in the future.
5. Synchronization Framework provides synchronization services for data between different devices via various transport layers /38/

Multimedia domain provides camera, imaging and video functionalities for the system. The following subsystems are included (Figure 24 in appendix 2):

1. Imaging and Video Adaptation offers specific codecs and containers for GStreamer (multimedia framework).

2. Camera Adaptation offers specific codecs and containers for GStreamer.
 3. UPnP, Universal Plug and Play provides stack and profile for UPnP audio and video.
 4. GStreamer provides playback, streaming, and imaging functionality to the device.
 5. Audio Adaptation provides PulseAudio specific modules.
 6. Pulse Audio provides a proxy between the hardware and the audio applications.
- /38/

Essentials domain provides system essential libraries and tools.

Communications domain provides the basic communication services. It has the following sub domains (Figure 24 in appendix 2):

1. IP Telephony, Instant Messaging and Presence enable real time communications via a pluggable protocol.
2. Cellular Framework provides cellular telephony stack and services. Multiple platform support.
3. Connection Manager provides management for internet connections.
4. The Bluetooth subsystem includes Linux Bluetooth stack BlueZ.
5. Communication Adaptation offers different plugins and modules. /38/

Qt domain includes different Qt toolkits. It has the following subsystems as seen in the Figure 24 in appendix 2:

1. Qt includes the application and toolkit.
2. Qt Mobility contains Qt Mobility and MeeGo APIs.
3. Qt WebKit provides web kit for web content rendering. /38/

Graphics domain enables the core graphics capabilities for the platform. The domain includes the following subsystems (Figure 24 in appendix 2):

1. Font Management includes a service for font locating and for selecting the fonts according to the requirements.
2. Input Adaptation includes the hardware behind drivers, Hardware buttons, qwerty keyboard, and touch screen are provided as input devices.
3. Implementation of the X11 Window system with specific drivers.
4. OpenGL ES provides OpenGL implementation and Khronos interfaces and specific implementations of certain drivers and libraries.

5. Display and Graphics Adaptation offers platform specific Framebuffer and display panel. /38/

Figure 25 displays a MeeGo API view, which includes Qt and Qt Mobility.

Qt provides functionality for the developers to build applications with graphical user interfaces.

Qt mobility provides the API's for the Qt. APIs allow developers to use features as cross platform. /38/

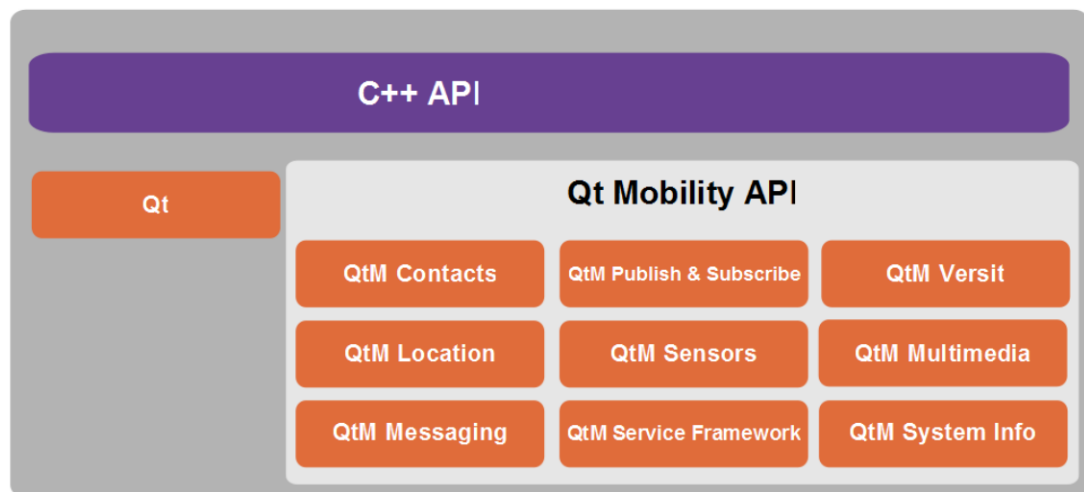


FIGURE 25: The Illustration of the MeeGo API view. /38/

MeeGo is using security framework called Mobile Simplified Security Framework (MSSF). It is the base of the MeeGo Security Architecture. Its purpose is to protect the owner's personal data and passwords and to prevent any device misuse. This can be accomplished with the following ways according to the MeeGo security architecture:

1. Access control framework that gives permissions to applications and also restrict them according to the application.
2. Integrity checking that verifies the executable authenticity.
3. Sensitive data protection is done by encryption and signing.

In all the other operating systems, the security main goals are:

1. Protecting the user from the hostile applications.
2. Protecting the device from the unwanted applications trying to change the parameters in the software.
3. Protecting the business by preventing SIM card simlocking functionality or the other important application functionality.

4. Enabling new services by enabling stronger security for example mobile payments. /70/

In MeeGo's Mobile Simplified Security Framework, the following components are in use:

1. Trusted Execution Environment Services (TEE) use secure cryptographic services for the other sub systems.
2. Integrity Protection makes sure that the device filesystem is safe.
3. Access Control is limiting the access to protected resources.
4. Cryptographic Services offers integrity and confidentiality protection.
5. Secure Software Distribution ensures that the software packages are authenticated properly and allows the security policy remote management.
6. Certificate Manager stores and manages security certificates and private keys.

The Figure 26 in appendix 3 shows how the different security components are situated and connected in the platform.

The Trusted Execution Environment, TEE is contained in the security network. The TEE and its subsystems provide interfaces implemented inside security hardware to prevent device tampering.

The following requirements form basis for the MSSF services on top of security hardware:

1. Securely Provisioned Root Public Key (RPK): A programmable key which is used to identify the SW stack.
2. Securely Provisioned Root Device Specific Key (RDSK): Used for guaranteeing the data confidentiality and integrity in cryptographic operations.
3. Trusted Execution Environment: The environment which ensures the cryptographic operations and key management are in the isolated environment. This is provided by the hardware. /70/

The access control in MeeGo ensures that the access to the operation system's resources is limited. A resource in MeeGo Access Control is virtual object that means some kind of functionality. Some of the resources are marked as sensitive because of the access they provide to some critical functions. The application needs to have specific credentials to access some of the resources. Linux standards offer the following

credentials: user identifier (UID), group identifier (GID), POSIX capabilities, and supplementary groups. MeeGo access control provides two additional credentials: Resource Token and Application Identifier. /70/

Resource token is a string that represents the access to the functionality. For example “Location” and “Cellular” are resource tokens. Cellular provides access to GSM call, SMS and GPRS data. D-Bus interfaces for csd-call, csd-sms, csd-gprs, and higher-level D-Bus interfaces are protected by this resource. Every application has possibility to define new tokens, which it provides and may want to use to protect its sensitive resources. When an application requests for resource token it needs to have a manifest file inside the package where it declares the credentials which it needs and provides. /70/

The Application Identifier is the other MeeGo credential. There are some requirements which the identifier needs to fill:

1. Identifier cannot be forged.
2. Identifier should identify the application.
3. Identifier should always remain the same. /70/

Package manager generates the identifier that meets the requirements. That is represented by the following:

AppID = {SourceID, PackageName, ApplicationSpecificName}

The SourceID is software source’s unique identifier. The PackageName is the name of the package. ApplicationSpecificName is defined by the developer. It is only provided when there are lots of different binaries that have the need for the different access rights. If it is not defined in the Manifest Files, it is assumed to be none. /70/

SMACK, mainline Linux Security Module (Simple Mandatory Access Control Kernel) provides mandatory access control model. The rules say that a subject can only access the objects with the matchable labels. The access rules are defined by Subject, Object, and Access. /70/

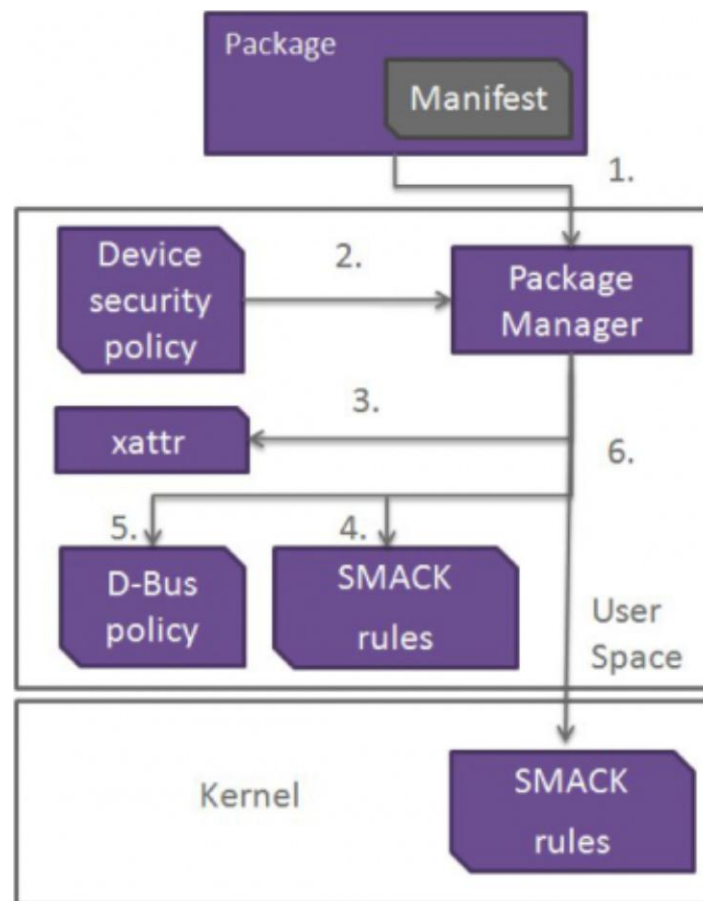


FIGURE 27: SMACK during the application installation /70/

When installing an application, the package and the Manifest Files come to the Package manager (1). (Figure 27) The manager verifies the device security policy to determine the amount of credentials to be granted (2). The package manager generates the identifier and sets the corresponding label in the attribute (3). Resource tokens and their access types are added to the SMACK rules by the Package Manager (4). If the application wants to protect D-Bus service it provides, the policy of the D-Bus configuration is updated (5). The additions are stored in the Kernel and the Package Manager updates them. (6). /70/

The access controls are enforced during the following ways according to the MeeGo security architecture Wiki:

1. Kernel Enforcement verifies that objects are protected with user identifiers (UID), group identifiers (GID), supplementary groups, and POSIX capabilities.
2. D-Bus Daemon is checking for credentials based on the D-Bus configuration policy.

3. Application Enforcement provides services for each application to use a certain library for credential checking. /70/

During the process execution, the process normally loads shared library to use some functionality. This makes it possible for a hostile program to get loaded to a process, which can then try to use the credentials. MeeGo uses a rule that while loading the libraries, the process should not own more credentials (UIDs, GIDs, supplementary groups, POSIX capabilities, and resource tokens) than the software source of the library is allowed to grant. This rule was in use in the MSSF version 1. However, version 2 uses LSM (Linux Security Module) composer. It allows combining the multiple LSM abilities into single entry. During the MeeGo 1.2 release, a composer was not available and because of that a new solution was added. The additional functionality was added directly in to the SMACK LSM using the mmap LSM hook to enforce the rule. When the application tries to load itself into a shared library, the SMACK LSM determines the maximum amount of credentials which are allowed to the process. /70/

The MSSF Integrity Protection subsystem verifies the integrity of all of the user space components, and anything that is in the filesystem. The MSSF version 1 integrity protection subsystem implementation was changed because mainline kernel already had its own integrity protection subsystem: Integrity Measurement Architecture (IMA). IMA uses the extended attributes for storing the reference hashes of files. That will allow the remove of the reference hashes from the Kernel memory during the system run-time. The reference cryptographic hash (SHA-1) is stored in the security.ima extended attribute every time the IMA module accesses the file; also its integrity gets verified. The security.ima extended attribute contains a non-keyed hash which results to a protection mechanism that gives no protection against offline modifications. The attacker is able to remove the storage and modify it in another system. For protection IMA uses an extension called the Extended Verification Module (EVM). It provides an offline protection for the Filesystem data, including the extended attributes. It calculates a keyed cryptographic hash (HMAC-SHA1) and stores it in the security.evm extended attribute. The calculation must be done by the trusted execution environment on the platform with hardware protected key. If it is not done this way the key should be stored somewhere in the Filesystem. This way there is no additional protection over IMA. /70/

The protection of the key can be done by the following two ways:

1. Direct usage of TEE services and keys:
 - Extended Verification Module (EVM) verifies the keyed cryptographic hash together with TEE (Trusted Execution Environment). TEE would provide the protection and the key would never be exposed outside the system. This could lead to performance issues because of the number of the operations.
2. Key export from TEE upon kernel validation:
 - After successful kernel integrity verification, the kernel is given a key. EVM module is then able to use the key to compute and verify the cryptographic hashes. However, this includes a risk of key compromise, because kernel will have a security defect that will be sooner or later discovered and made exploitable. This method does not have too serious effect on the performance because every operation is not invoked by the subsystem. /70/

The MSSF Cryptographic Services subsystem provides additional way to ensure system integrity and confidentiality. The services are provided for the following interfaces:

1. Local cryptographic services
2. Key management. /70/

The protected data can be accessed in two ways:

1. With the application specific key: It can be derived from the application identifier.
2. With the shared key: It can be derived from the resource token and RDSK key (device path). /70/

The software distribution is linked to the security architecture. The developers and the users can trust that the software has originated from authenticated sources. The MeeGo package source can be either known or unknown. Each of the known software sources has the level of trust defined. The package manager has the access to the list and the public keys. The software update rules are defined by the trust level. The package can be only updated from the source it was originally installed or from a software source with a higher trust level. This means that the trusted software cannot be updated from a

less trusted source. The software source information is preserved through the application lifetime. /70/

7. SYSTEM UNDER TESTING

This chapter tells about the testing and the actual investigation of the system under testing. The actual testing theory is explained in the chapter 3. This chapter focuses more on the automaton side because that had more problems and variations with the results. The idea of the project was to test the phone modem software.

This testing project began on my account about three years ago when I started in the test automation project. I had never been in this kind of project before. When this testing began, there were four testers and four different computer/device combinations. At this point there are only two testers and three computers in use. The computers in use are introduced next. There were two large tower PC's and one laptop. All of the computers were recycled and not new when they were taken in to use in the project.

Computer one was a normal desktop PC, Dell Precision 380 with Intel 3.0GHz P4, 2GB of RAM. The computer had ATI Radeon Firepro 3D V3750 256MB PCIe graphics card, with Windows XP Service Pack 2 installed.



FIGURE 28: Desktop PC 1 /46/

The second computer was a Lenovo ThinkPad T60 laptop with Intel T2500 2.0 GHz Core Duo processor. Laptop had 1GB RAM and ATI X1400 128MB graphics card and Windows 7.



FIGURE 29: Laptop computer 2 /47/

The third computer was Dell Optiplex Gx280 Tower with Intel Pentium 4 2.8GHz processor, 1GB of RAM. The computer had integrated Intel Extreme Graphics graphics card.



FIGURE 30: Desktop PC 3 /48/

The problem was that every computer used a little bit different tricks for ATS test sets. The flashing operation basics were the same with all the computers. Flashing was done using two FPS-21 flashing devices. Flashing software versions varied also. There were two devices in use. Software versions did not get changed except when it was absolutely needed. This happened when the phone could not be flashed any more.

The flashing devices were updated through installing the flashing software. If either of the flashing program software or the device software was too old, the device was not

recognised and the device could not be flashed. Both of the flashing devices were using the same software to minimize the flashing problems. Generally, it was a good idea not to update anything unless absolutely necessary. The flashing program was different version in all of the computers (A little more about the flashing software is explained in the chapter “Testing tools in general”).

The biggest problem was that when everybody flashed the same software to the mobile device, the automated tests did not work with everybody. This automated testing project followed black-box testing rules. As explained in the previous chapters, black box tester does not be aware of what is happening inside the device. Only the inputs and outputs are known, that's why the different results did not cause that many problems any more. The wanted results were known in advance.

The testing used the test setup in the Figure 31. In the Figure, a mobile phone is attached to a flashing device with either USB or Fbus cable. Flashing device (FPS-21) is attached to the network. Computer is in the ATS network and it is used to monitor the test runs. The device under testing can get the power from the Fbus cable (from the flashing device) or from the outside power source (less than 4V). The power source in use was ELC DC Power Supply.

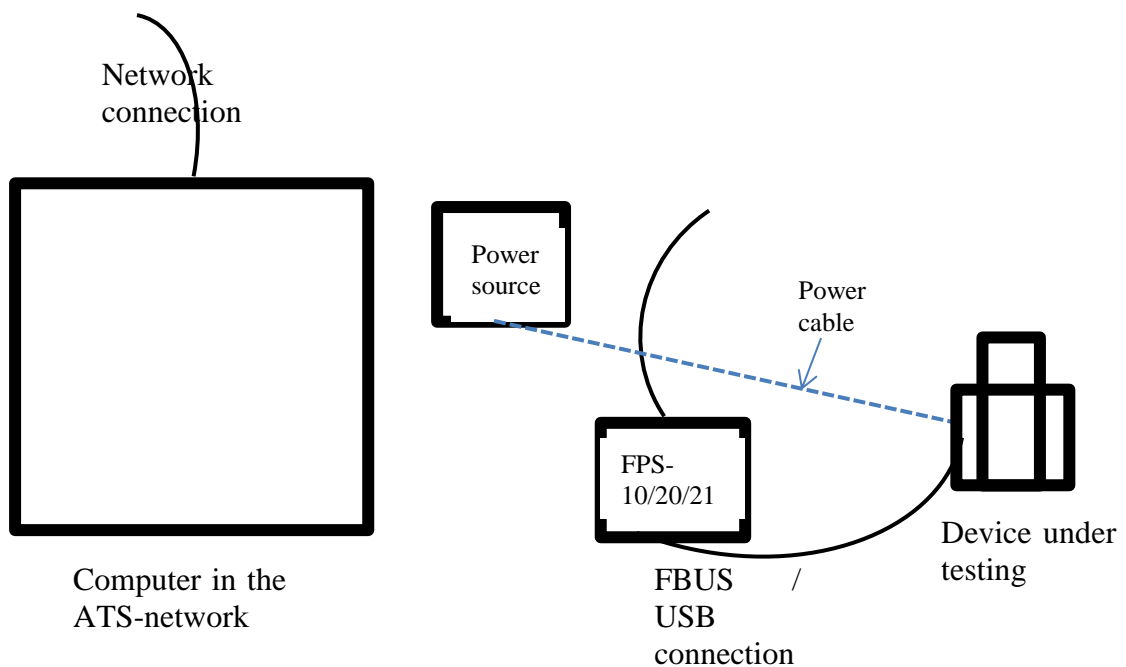


FIGURE 31: Test setup

As explained before, every computer system created a little different test sets to ATS causing confusion. Tests could also be started as a rerun from the ATS interface. In that case, the test runs started normally. The sets were fetched from the testing software using a piece of code called “the spell”. They were formed a little bit differently with each of the computers. If the code worked with one of the testers, it did not necessarily work with any of the others. Some of the mobile devices worked when the power came from the power source. Flashing device provided also the power but it was not as reliable as the power source which delivered exactly the power that was needed.

“The Spells” used Perl and they looked like this: “Perl “Perl program” “Name of the worker machine” -atsserver -p “Test script path” -s “Test set path ”\\” -decode -a -priority 2 -z -dis. It is Perl script and a user has to give a worker machine name, which was usually user dedicated PC. After that comes the name of the used ATS server and the test script path. The scripts that included the automated tests were stored in the software B (“an adaptation” between modem software and phone user interface software.). The sets were fetched from the test program Mercury Quality Center. Test set path tells the ATS where to get the sets. The system knows then if some tests are missing from scripts and they are not executed. See Figure 32 for the idea, where the software is “situated”. Test scripts are included in the “software B”.

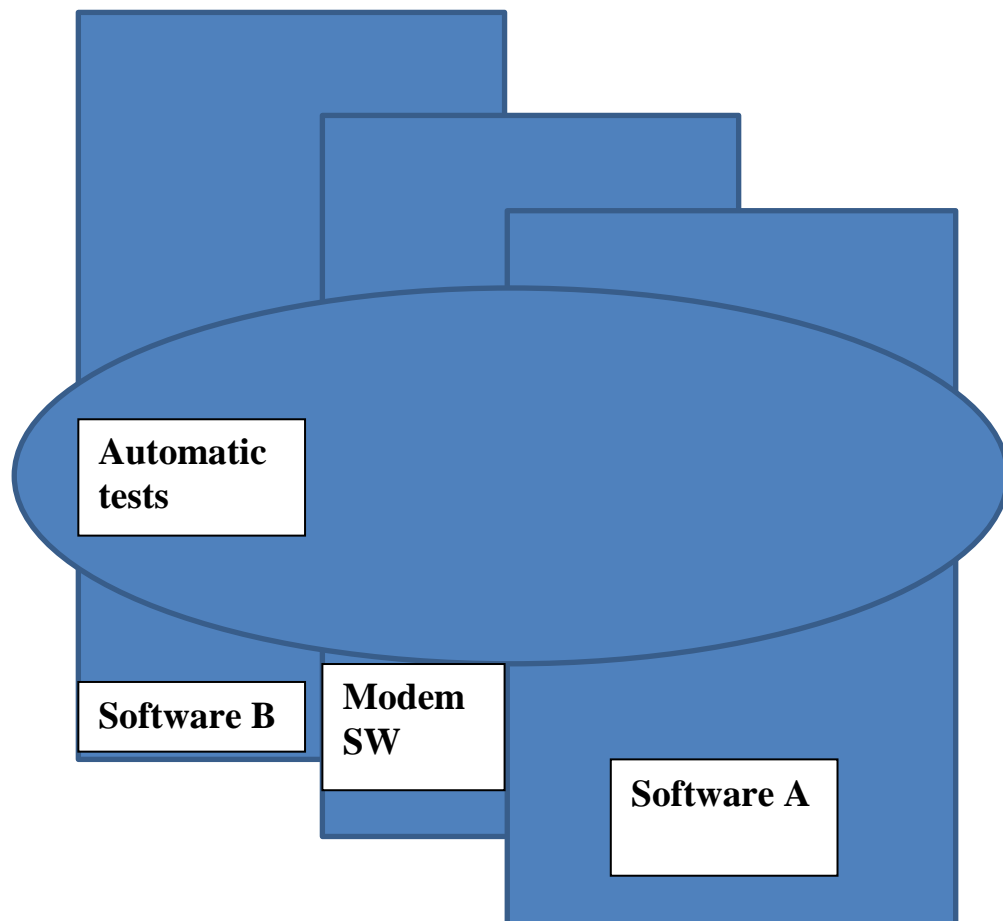


FIGURE 32: “software A and B” /12/

The first testing started with installing the modem software, the software under testing. Normally it was enough to flash only software (modem software) to the device, instead of both software A and B. The same software was flashed to all the mobile devices. There can sometimes be differences between the mobile device versions. The older version of the mobile device was only in use if there were no newer available or the one did not work. There was always software A and B in the phone before flashing modem software. Only when the mobile device was “brand new”, it was “empty” or it had the user interface. Normally there was not a need to flash anything but the software under testing which was the fastest situation.

There were sim cards with three different operators in use. Both of the testers had one Elisa, two Sonera and one DNA sim cards. It was never really known why but one Sonera sim card especially gave error code -4159, which meant “KerrGprsActivationRejected”, can't connect to the internet. In some phones, sim card's contact surface did not touch the phone properly which seemed to be the number one cause for the error -4159. That happened especially with Sonera sim cards. Sometimes

these problems were solved by pressing the sim card, in order that the contact surface had contact with the phone. (Symbian error codes can be seen at http://www.developer.nokia.com/Community/Wiki/Symbian_OS_Error_Codes).

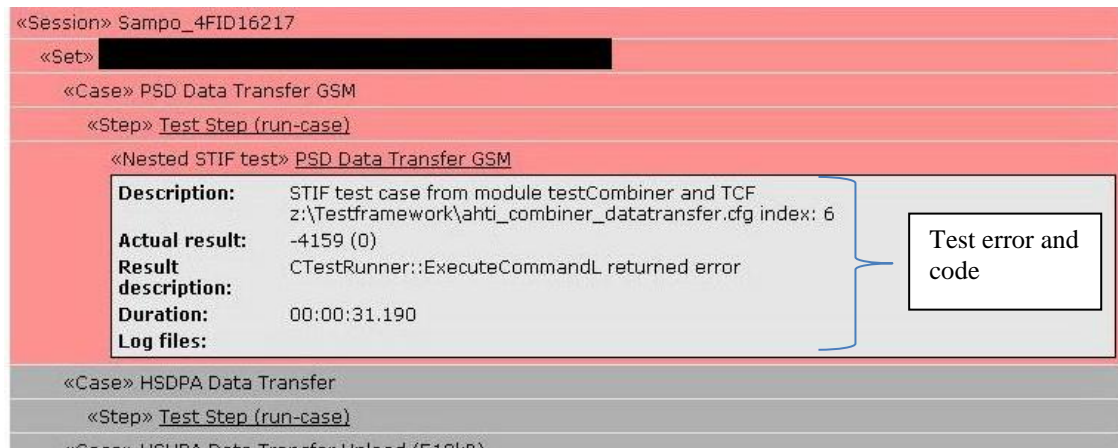


FIGURE 33: Error code -4159

Sometimes during testing, at least with one of the devices, the ATS testing process had been started but it resulted in “finished” in the middle of the execution (Figure 34). This was not an error but the test stopped for some reason (a time out). One of the reasons and usually the first to check was that Phoenix program was not closed. Phoenix program reserved the same connection than the test runs. The other reason was that Fbus cable was not connected.

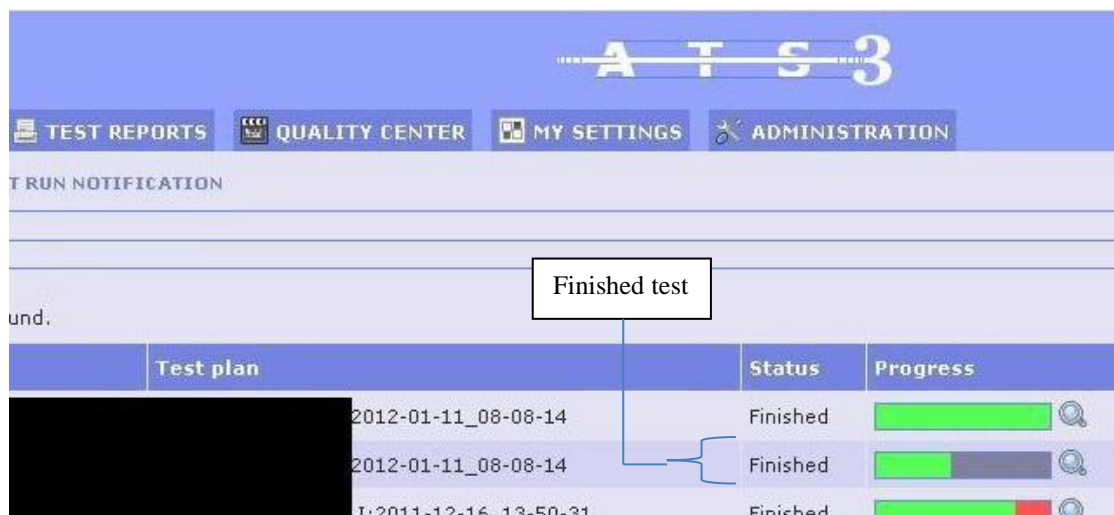


FIGURE 34: Finished test.

When it was certain that nothing was reserving the connection and the sim card was properly attached, the tests were still not always finishing. In addition to the modem

software testing, we also tested the software A with the different test cases. That way we could eliminate whether the problems were really caused by the modem software. Sometimes the software A or B were really causing the problems.

If nothing previously listed ways worked, then the prior week's modem software was flashed again. If that worked, and the software A and B were the same in both cases, the current software under testing was faulty. But if that still did not help, the software A and B needed to be flashed again and start to trace the problem. Since the software A was tested weekly, it was clear if it was “broken”. That helped to minimize the efforts in finding the problem. Usually that helped and the test was finished correctly.

In the end, it was never found out why the testing results sometimes differed so much. At the time when we had lots of problems in the project with this issue, there were three to four testers. And if the device did not work with one tester he/she gave it to the other one. There was always one who could get the test completed. This project had been going on for many years so it was pretty much known when the test results were not what we expected. These problems really lost their meaning because the project ended at the end of August. In addition, due to the fast downgrade of the Symbian platform, the operating system has lost its importance.

REFERENCES

1. Maarit Kopra, ATS3-järjestelmän evaluointi. 26.2.2007. Read 13.8.2012
<http://www.doria.fi/bitstream/handle/10024/4278/stadia-1174291287-9.pdf?sequence=1>
2. Linux. Wikipedia. 23.7. 2012. Read 09.08.2012
<http://en.wikipedia.org/wiki/Linux>
3. MeeGo. Wikipedia 01.08.2012. Read 09.08.2012
<http://fi.wikipedia.org/wiki/MeeGo>
4. MeeGo. Wikipedia 01.08.2012. Read 09.08.2012
<http://en.wikipedia.org/wiki/MeeGo>
5. Haikala, Ilkka & Märijärvi, Jukka, Ohjelmistotuotanto. 7.painos. Satku Kauppakaari Oyj, Pieksämäki 2001.
6. Test case. Wikipedia 21.07.2012. Read 13.08.2012
http://en.wikipedia.org/wiki/Test_case
7. Heli Kontturi. 2005. FIM-selaintyökalun testaus
8. Black-box testing. Wikipedia 11.08.2012. Read 14.08.2012
http://en.wikipedia.org/wiki/Black-box_testing
9. White-box testing. Wikipedia 07.08.2012. Read 14.08.2012
http://en.wikipedia.org/wiki/White-box_testing
10. "Testing A Software" World: Development model: V-model for testing 17.11.2011. Read 16.08.2012 <http://testing-a-software.blogspot.fi/2011/11/development-model-v-model.html>
11. "Testing A Software" World: The Earlier Bug Is Found Cheaper It Costs 03.11.2011. Read 16.08.2012 <http://testing-a-software.blogspot.fi/2011/11/earlier-bug-is-found-cheaper-it-costs.html>
12. Discussions with the colleagues in 2012.
13. Software Testing Fundamentals: Software Testing Levels 2012. Read 17.8.2012
<http://softwaretestingfundamentals.com/software-testing-levels/>
14. Gray-box testing. Wikipedia 11.08.2012. Read 19.08.2012
http://en.wikipedia.org/wiki/Gray_box_testing
15. Decision table. Wikipedia 15.06.2012. Read 20.08.2012
http://en.wikipedia.org/wiki/Decision_table
16. All-pairs testing . Wikipedia 02.07.2012. Read 20.08.2012
http://en.wikipedia.org/wiki/All-pairs_testing

17. State transition table. Wikipedia 06.11.2011. Read 20.08.2012
http://en.wikipedia.org/wiki/State_transition_table
18. Control flow. Wikipedia 16.08.2012. Read 20.08.2012
http://en.wikipedia.org/wiki/Control_flow
19. Mark New. Data flow testing. Read 20.08.2012
<http://www.cs.swan.ac.uk/~csmarkus/CS339/dissertations/NewM.pdf>
20. Jkinfoline. What is White Box Testing(Glass Box Testing) ?
2010. Read 20.08.2012 <http://www.jkinfoline.com/white-box-testing.html>
21. Ian Sommerville. Path testing. 2008. Read 20.08.2012 <http://www.cs.st-andrews.ac.uk/~ifs/Books/SE9/Web/Testing/PathTest.html>
22. Code coverage. Wikipedia 18.08.2012. Read 20.08.2012
http://en.wikipedia.org/wiki/Code_coverage
23. Software Testing Mentor. Decision coverage or Branch Coverage. 2010. Read
20.08.2012 <http://www.softwaretestingmentor.com/test-design-techniques/decision-coverage.php>
24. Alex Samurin geocities.com/xtremetesting. July 2003. /
eXtremeSoftwareTesting.com. 2009.
<http://extremesoftwaretesting.com/Articles/WorldofGrayBoxTesting.html>
25. Manual testing. Wikipedia 31.07.2012. Read 21.08.2012
http://en.wikipedia.org/wiki/Manual_testing
26. Waterfall model. Wikipedia 08.08.2012. Read 21.08.2012
http://en.wikipedia.org/wiki/Waterfall_model
27. Agile software development. Wikipedia 13.08.2012. Read 22.08.2012
http://en.wikipedia.org/wiki/Agile_software_development
28. Test automation. Wikipedia 21.08.2012. Read 21.08.2012
http://en.wikipedia.org/wiki/Test_automation
29. Dynamic testing. Wikipedia 21.08.2012. Read 22.08.2012
http://en.wikipedia.org/wiki/Dynamic_testing
30. Static testing. Wikipedia 24.03.2012. Read 22.08.2012
http://en.wikipedia.org/wiki/Static_testing
31. Functional testing. Wikipedia 08.00.2012. Read 22.08.2012
http://en.wikipedia.org/wiki/Functional_testing
32. Non-functional testing. Wikipedia 20.01.2012. Read 22.08.2012
http://en.wikipedia.org/wiki/Non-functional_testing

33. Usability testing. Wikipedia 17.07.2012. Read 22.08.2012
http://en.wikipedia.org/wiki/Usability_testing
34. Graphical user interface. Wikipedia 29.05.2012. Read 22.08.2012
http://en.wikipedia.org/wiki/GUI_testing
35. Software testing. Wikipedia 17.08.2012. Read 22.08.2012
http://en.wikipedia.org/wiki/Software_testing
36. Software Testing Stuff Read 24.08.2012
<http://www.testingstuff.com/tools.html>
37. Bridgette Townsend. Kibilogic.com. Taking a Closer Look At MeeGo. 26.06.2010. Read 25.8.2012 <http://kibilogic.com/wordpress/wp-content/uploads/2010/11/>
38. MeeGo Architecture. 29.9.2010 Read 21.9.2012 <https://meego.com>
39. Nokia N9. Wikipedia. 17.9.2012. Read 20.0.2012.
http://en.wikipedia.org/wiki/Nokia_N9
40. Symbian Foundation Limited. 2010. HTI. Read 25.08.2012
<http://www.symbian.org/mirror/developer.symbian.org/wiki/HTIhtml.html>
41. Apache Tomcat. Wikipedia 23.08.2012. Read 25.08.2012
http://en.wikipedia.org/wiki/Apache_Tomcat
42. MySQL. Wikipedia 22.08.2012. Read 25.08.2012
<http://en.wikipedia.org/wiki/MySQL>
43. Sourceforge. Software Testing Automation Framework. 29.06.2012. Read 25.08.2012
<http://staf.sourceforge.net/>
44. Perl. Wikipedia 22.08.2012. Read 25.08.2012 <http://en.wikipedia.org/wiki/Perl>
45. Fbus Wikipedia 10.10.2011. Read 27.08.2012 <http://en.wikipedia.org/wiki/FBus>
46. ServerMonkey.com. 2012. <http://www.servermonkey.com/dell-precision-380-workstation-3-0ghz-2gb-80gb-dual-monitor-graphics/>
47. TechTarget, 2000-2012. Read 5.09.2012
<http://www.notebookreview.com/default.asp?newsID=2767>
48. Free Full Registered Software, Read 5.09.2012.
<http://www.faheemqureshi.blogspot.fi/2012/03/download-drivers-for-dell-optiplex.html>
49. Nokia Phoenix Service Software-CKB-Cell Phone Knowledge Base. Wikipedia 12.08.2012. Read 09.09.2012
http://www.cpkb.org/wiki/Nokia_Phoenix_Service_Software

50. List of tools for static code analysis. Wikipedia 08.09.2012. Read 10.09.2012
http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis
51. List of tools for dynamic code analysis. Wikipedia 10.09.2012. Read 10.09.2012
http://en.wikipedia.org/wiki/Dynamic_code_analysis
52. Selenium – Web browser Automation. Read 11.09.2012 <http://seleniumhq.org/>
53. Model–view–controller. Wikipedia 11.09.2012. Read 11.09.2012
<http://en.wikipedia.org/wiki/Model-view-controller>
54. Model-View-Controller. Microsoft. 2012. Read 12.09.2012.
<http://msdn.microsoft.com/en-us/library/ff649643.aspx>
55. LabView. Wikipedia 28.08.2012. Read 12.09.2012
<http://en.wikipedia.org/wiki/LabVIEW>
56. IBM Rational Functional Tester. Wikipedia 22.08.2012. Read 12.09.2012
http://en.wikipedia.org/wiki/IBM_Rational_Functional_Tester
57. Selenium_(software). Wikipedia 09.09.2012. Read 12.09.2012
http://en.wikipedia.org/wiki/Selenium_%28software%29
58. Visual_Studio_Test_Professional. Wikipedia 06.09.2012. Read 12.09.2012
http://en.wikipedia.org/wiki/Visual_Studio_Test_Professional
59. History_of_Symbian. Wikipedia 25.08.2012. Read 13.09.2012
http://en.wikipedia.org/wiki/History_of_Symbian
60. Symbian. Wikipedia 10.09.2012. Read 13.09.2012
<http://en.wikipedia.org/wiki/Symbian>
61. Testing Geek. Atlantis Software. 2012. <http://www.testinggeek.com/mercury-quality-centre-introduction>
62. Marco Aiello. 2005. Introduction to Symbian OS. [PDF-document]
63. Lance Li. 2007. Symbian OS Architecture. [PDF-document]
64. Symbian OS. Nokia Developer Wiki. Nokia. 26.7.2012. Read 19.9.2012
http://www.developer.nokia.com/Community/Wiki/Symbian_OS
65. Heise Media UK Ltd. 2010. Hacker plants back door in Symbian firmware
<http://www.h-online.com/security/news/item/Hacker-plants-back-door-in-Symbian-firmware-1149926.html>
66. Jane Sales. 2005. Symbian OS Internals. Real-time Kernel Programming. Wiley Publishing, Inc. [PDF-document]

67. UID Q&As (Symbian Signed). Nokia Developer Wiki. Nokia. 26.7.2012. Read 19.9.2012.
http://www.developer.nokia.com/Community/Wiki/UID_Q%26As_%28Symbian_Signed%29
68. Andreas Jakl. 2008. Symbian OS 9, Platform Security. [PDF-document]
69. Nokia_808_PureView. Wikipedia 10.09.2012. Read 21.09.2012
http://en.wikipedia.org/wiki/Nokia_808_PureView
70. The Linux Fiundation. 27 .06.2011. Read 22.09.2012
<http://wiki.meego.com/Security/Architecture>

APPENDICES

Appendix 1 FIGURE 17: Symbian kernel /63/

Appendix 2 FIGURE 24: MeeGo domain view /38/

Appendix 3 FIGURE 26: The MeeGo security Architecture /70/

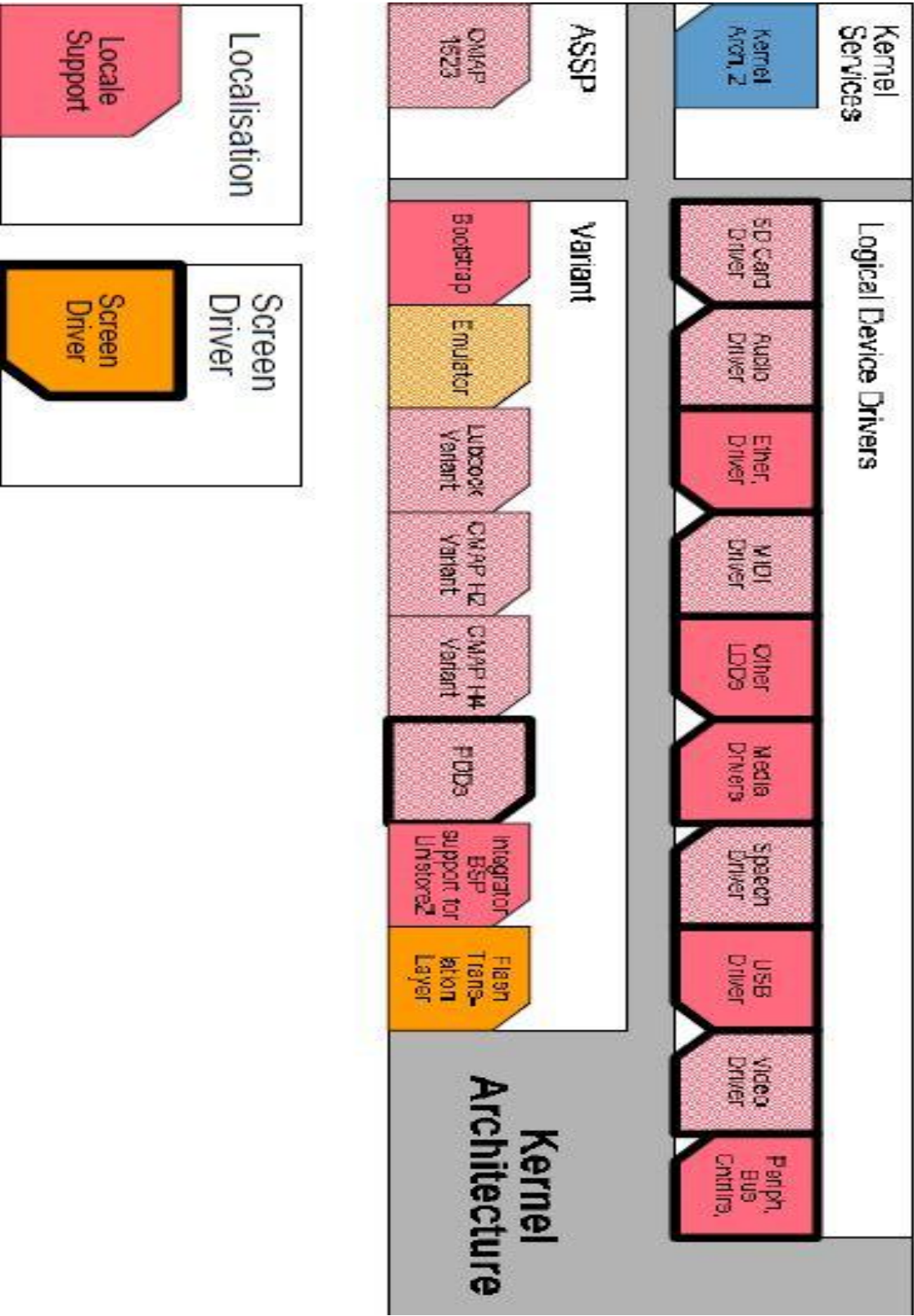


FIGURE 17: Symbian kernel /63/



FIGURE 24: MeeGo domain view /38/

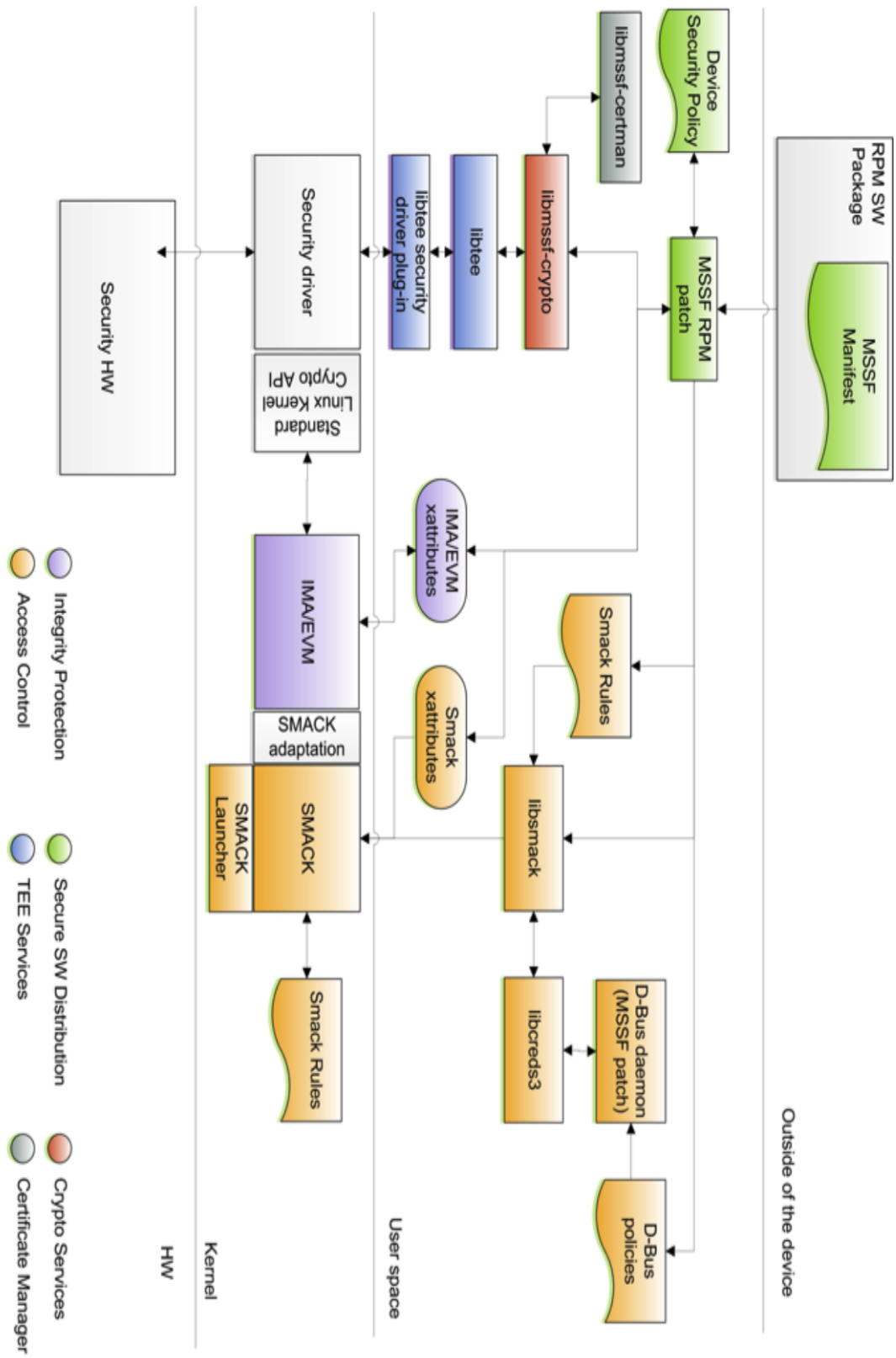


FIGURE 26: The MeeGo security Architecture /70/